修 士 論 文

STS プロトコルの形式化と検証による CafeOBJ と Coq の比較

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

原 光太朗

2005 年 3 月

修 士 論 文

STSプロトコルの形式化と検証による CafeOBJとCoqの比較

指導教官　片山卓也 教授

審査委員主査　片山卓也 教授
審査委員　落水浩一郎 教授
審査委員　鈴木正人 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

310088 原 光太朗

提出年月: 2005 年 2 月

概 要

セキュリティプロトコルの解析と検証を行う方法としては形式手法が有効であると考えられている. 形式手法はシステムを数学的にモデル化し形式仕様の作成および仕様の検証を行う手法で, 厳密性・無矛盾性の点で優れており, これまでに多くの方法が提案されている. しかし方法の提案や実例を扱う研究に比べ, その比較を扱う研究はあまり行われておらず, その結果, 形式手法を用いる際のガイドラインがないという問題点が存在する.

本論文ではセキュリティプロトコルの一つである STS プロトコルを例題として取り上げ, 安全性と信頼性を有しているかどうかの検証を, 異なる二つの形式手法として CafeOBJ を用いる方法と Coq を用いる方法の双方で行う. この具体例を通して, 双方によるシステムの形式化および検証の手法についての比較を行い, 長所・短所を明らかにすることにより形式手法に関する新たな指針を示すことを目指す.

目 次

第1章　序論

インターネットに代表される広域情報ネットワークの急速な普及および発展に伴い, セキュリティプロトコルを始めとした通信プロトコルの研究が広く行われている. セキュリティプロトコルは情報を暗号化し通信者間で秘密の通信を行うためのプロトコルで, 近年のネットワークセキュリティへの関心の高まりとともに盛んに考案が行われている. 交通システムや金融システムなど多くのシステムがネットワークを介して重要な情報のやり取りを行っている現在, 期待した通りの情報のやり取りが安全に行われるなどのセキュリティプロトコルの正しさを保証することの重要性が増してきている.

セキュリティプロトコルの解析と検証を行う方法としてはNSPK プロトコルの不具合が報告されたこと [8] などから, 形式手法が有効であると考えられている. 形式手法による検証は, システムを数学的にモデル化し形式仕様の作成および仕様の検証を行う技法で, 厳密性・無矛盾性の点で優れておりシステムの高信頼性に効果が高い. このようなことから, セキュリティプロトコルの正しさを形式的に検証するための技法が広く研究され, 多くの方法が提案されている. 代表的なものに, 代数仕様言語を用いる方法 [3, 4, 5], 証明支援系を用いる方法 [9, 10], 様相論理を用いる方法 [11], モデル検査を用いる方法 [12, 13] がある.

ここに一つの問題点が存在する. その問題点とは, それぞれの手法は独自の理論や論理体系に基づいており形式化や検証法は個々の形式手法に依存した特徴を持ったものになっているにも関わらず, 方法の提案や実例を扱う研究に比べその比較を扱う研究はあまり行われていないということである. その結果, 形式手法を用いる際のガイドラインがなく, 目的によって手法を使い分けたり併用したりすることができない.

そこで本研究では, セキュリティプロトコルの一つである STS プロトコル [1] を例題として取り上げ, 正しさを保証する性質として安全性と信頼性を有していることを異なる二つの形式手法により検証する. 安全性と信頼性を示す性質として, 不正を働く主体のなりすまし等による攻撃によって誤った認証鍵の交換が行われないこと, およびプロトコルの正規の参加者である主体同士が正しく認証鍵を交換することを確かめる. 異なる二つの形式手法として, 等式論理に基づく代数仕様言語 CafeOBJ[17] を用いる方法と高階論理に基づく証明支援系 Coq[18] を用いる方法の二つを用いる.

仕様記述を行う際に必要となるシステムのモデル化には双方とも観測遷移機械 OTS (Observational Transition System) を用いる. OTS ではシステムがどのように振る舞うかを観察する. すなわち, システムに関連する値が実行に伴いどのように変化するのかを観測することで, システムのモデルを作成する.

本研究における STS プロトコルの形式手法による検証の流れを以下に示す. まず, STS プロトコルのモデルを OTS を用いて作成する. 次に, 作成した STS プロトコルのモデルを CafeOBJ と Coq それぞれで記述することによって形式化し, STS プロトコルの CafeOBJ 仕様および Coq 仕様を作成する. 続いて, 安全性と信頼性を表す性質を同じく CafeOBJ と Coq それぞれで記述する. 最後に, その記述が正しいことを双方の処理系で示す[1] ことによって仕様の検証を行う.

本研究ではさらにこの STS プロトコルを対象とした形式手法による検証の具体例を通して, CafeOBJ と Coq 双方によるシステムの形式化および検証の手法についての比較を行う. 具体的にはモデルの記述法や処理系における検証などに焦点を当て, 特徴を捉えることによって比較を行うことにより, CafeOBJ と Coq それぞれを用いる形式手法の長所・短所を明らかにする. さらに, 考察を加えることでシステムの正しさを保証することを目的とする形式手法に関しての新たな指針を示す.

本論文の以下の構成は次の通りである. まず, 形式手法として用いる CafeOBJ と Coq についてそれぞれ 2 章, 3 章で触れる. 続いて, 4 章でシステムのモデル化に用いる観測遷移機械について触れ, 5 章で例題とする STS プロトコルについて触れる. その後, 6 章でそれぞれの手法による例題プロトコルの形式化 (仕様作成) を, 7 章で仕様の検証を行い, 8 章でそれに関する考察を行う. 9 章で関連研究に触れ, 10 章で結論を述べる.

[1] 以後, この過程および結果を特に「証明」もしくは「証明する」と言う.

第2章　代数仕様言語CafeOBJ

代数仕様言語 CafeOBJ[14] は, モジュールという記述単位を取って等式仕様および振舞仕様を記述する仕様記述言語である.

等式仕様は主に自然数やスタックなどの抽象データ型の記述に用いる. 例として, 自然数の構造を表すモジュール NAT と, 群を表すモジュール GROUP の記述を以下に示す.

```
module! NAT {
  [Nat]
  op 0 : -> Nat
  op s _ : Nat -> Nat
}
```

```
module* GROUP {
  [G]
  op 0 : -> G
  op _+_ : G G -> G   op -_ : G -> G
  vars X Y : G
  eq 0 + X = X .
  ceq Y + X = 0 if Y = (-X) .
}
```

一つのモジュールの記述は,

module(mod) モジュール名{

で書き出しが始まり, 対応する}でモジュールの記述が終わる. モジュールはシグネチャと公理の二つの部分から成る. シグネチャ部はプログラミング言語の型に相当するソートと演算子の宣言を行う指標である. 一方, 公理部では等式を宣言することによって演算の定義を行う.

モジュールの記述には固いものと緩いものの二つがある. 固いモジュールで表される集合に属する要素は, 記述されたシグネチャと公理以外には他のものを含まない. 一方, 緩いモジュールで表される集合に属する要素は, 記述されたシグネチャと公理を満たしていれば, 他のものを含んでいてもよい. 固いモジュールには module の後に'!' を付け, 緩いモジュールには'*' を付ける. NAT は固いモジュールであり, GROUP は緩いモジュールである.

ソートは'[' と']' で囲って宣言する. 演算は'op' で宣言する. 'op' の後には, 演算子を記述し, ':' と引数のソート列を書く. 最後に'->' と結果のソートを書く. 例えば, モジュール NAT においては自然数を表すソート Nat を宣言し, Nat である 0, および Nat である引数を一つ取り結果として Nat であるものを返す s という二つの構成子である演算を宣言することによって自然数の構造を表現している.

等式の宣言は'eq'で開始し, 条件付き等式の宣言は'ceq'で開始する. 'eq'の後は左辺式と右辺式を'='で連結し, 最後に'.'を付ける. 'ceq'の後は左辺式と右辺式を'='で連結し, 続いて'if'および条件式を記述し, 最後に'.'を付ける. 例えば, モジュール GROUP においては「単位元 0 と任意の元 X の和は X である」ことを一つ目の等式で, 「元 Y が任意の元 X の逆元であるなら, X と Y の和は単位元 0 である」ことを二つ目の条件付き等式で表現している.

一方, 振舞仕様は抽象機械の記述に用いる. 例として, 抽象機械として捉えた整数のリストを表すモジュール LIST の記述を以下に示す.

```
module* LIST {
  pr(NAT) *[List]*
  op err : -> ?Nat
  op nil : -> List            bop cons : Nat List -> List
  bop car : List -> ?Nat      bop cdr : List -> List
  var N : Nat  List L : List
  eq car(nil) = err .         eq car(cons(N, L)) = N .
  eq cdr(nil) = nil .         eq cdr(cons(N, L)) = L .
}
```

モジュールには組み込みのモジュールや定義されたモジュールを輸入することができる. 輸入されるモジュールに変更がない場合には'protecting(pr)'を使用する. モジュール LIST においては, モジュール NAT を輸入し, ソート Nat および二つの構成子を輸入している.

振舞仕様ではソートは二種類に分けられる. 抽象データ型を表す可視ソート (visible sort) と抽象機械の状態空間を表す隠蔽ソート (hidden sort) である. 可視ソートは等式仕様におけるソートと同じものである. 隠蔽ソートに関しては二種類の演算があり, 状態を変化させる作用演算 (action operation) と状態を観察する観測演算 (observation operation) がそれである. 作用演算は抽象機械の状態と 0 個以上のデータを引数にとり, 抽象機械の変化後の状態を返す. 観測演算は抽象機械の状態と 0 個以上のデータを引数にとり, その状態に関連する値を返す. 基本的にこれらの演算の定義は, 作用演算による状態の変化によって各観測演算の値がどのように変化するのかを示す等式で表される.

隠蔽ソートは'*['と']*'で囲って宣言する. 作用演算と観測演算の宣言は'bop'で始め, その後には演算子を記述し, ':'と引数のソート列を書く. 最後に'->'と結果のソートを書く. モジュール LIST においては隠蔽ソート List を宣言し, 作用演算としてリストを形成する演算 cons を, 観測演算としてリストの先頭の要素を返す演算 car およびそれ以外の部分を返す演算 cdr を宣言している. また, 等式を用いてそれらの演算の定義を行っている.

CafeOBJ の実装として CafeOBJ 処理系 [15] がある. CafeOBJ 処理系は記述された仕様の等式を左から右への書き換え規則とみなし, 与えられた項を書き換える (簡約する) ことができる. この実行可能性により, 仕様記述の段階でシステムの模擬実験を行ったり, システムがある性質を有することの検証を行うことができる.

第3章　定理証明支援器Coq

定理証明支援器 Coq[18] は主に高階論理 (higher order logic) に基づく証明支援系である. Coq では集合や述語を帰納的に定義することができ, かつ帰納的定義を行うとそれらに付随する帰納法の原理等が自動的に生成される.

例えば, 自然数の集合 nat は以下のように帰納的に定義されている.

```
Inductive nat : Set := O : nat | S : nat -> nat.  (1)
```

一般に帰納的集合の定義は以下のように記述される.

```
Inductive 集合名 : Set := コンストラクタ : 型 | ... | コンストラクタ : 型.
```

(1) 式が定義することは以下の通りである.

- O は nat である.
- n が nat であるとき, (S n) は nat である.
- 集合 nat は以上のみで定義される.

この定義により, nat の構成に関する帰納法の原理が自動的に生成される.

述語も同様に帰納的に定義できる. 例えば, 自然数 n が偶数であることを示す述語 even は,

```
Inductive even : nat -> Prop :=
  even_0 : (even O)
| even_n : (n:nat)(even n) -> (even (S (S n))).  (2)
```

と定義できる. 帰納的述語の定義の記述は,

```
Inductive 述語名 : 引数の型 -> Prop :=
```

で始まる. 引数を複数取るものは引数の型を'->' で連結することで表す. 続いてその述語の定義となるものを' 証明名 : スキーマ' の形で記述する. 複数ある場合は'|' を用いて並べ, '.' で記述を終える. ここでスキーマとは, 変数の宣言とその変数に関係した述語を一緒にグループ化したものである.

(2) 式が定義することは以下の通りである.

- (even 0) は真である. その証明の名は even_0 である.
- (even n) が真であるとき, (even (S (S n))) は真である.
  その証明の名は even_n である.

この定義により, 述語 even の導出に関する帰納法の原理が自動的に生成される. また, スキーマの記述が正しいことを示す even_n などの証明は, スキーマの内容を持った推論規則 (公理系) として検証時に用いられる.

帰納的集合や述語を利用し, ラムダ記法で「項」に「名前」を与えることで新たな定義を行うこともできる. その定義は、

```
Definition 項名 [引数] (: 型) := 項.
```

の形で記述する. 例えば, 自然数 n が奇数であることを示す述語 odd は,

```
Definition odd[n:nat] : Prop := ~(even n).
```

と定義でき, 与えられた自然数が 0 なら true を, それ以外なら false を返す関数 jzero は、

```
Definition jzero[n:nat] := Cases n of 0 => true | (S _) => false end.
```

と定義できる.

以上の定義では, 多相型の考えを用いて型変数をパラメタとして使用することにより, 汎用のリスト等を定義できる. また, 性質や条件を表す述語 (スキーマ) に, 関数や述語に関する量化が可能である[1].

定理等の性質の記述は「名前」に「型」としてスキーマを与えることで記述できる. この記述は,

```
Theorem(Lemma) 定理 (補題) 名 : スキーマ.
```

の形で行われる. 例えば, 自然数 n は偶数か奇数のどちらかであるという定理は,

```
Theorem e_or_o : (n:nat) {(even n)}+{(odd n)}.
```

と記述できる.

Coq での検証には Coq の証明モードを用いる. 性質を示す定理等の入力を行うと自動的にこのモードに移行する. ユーザーはタクティクと呼ばれるコマンドの入力により, 定理であるスキーマの簡約を行う形で検証を進める. 集合や述語の帰納的定義を行う際, 新たに生成・導出される帰納法の原理や推論規則により, Coq の論理は拡張され, 証明モードにおける帰納法や場合分けは自動化される. そのためユーザーは検証を半自動的に行うことができる.

[1] 高階述語論理. 例えば,(EX n:nat | (even n)) は偶数の自然数が存在することを表す.

第4章　観測遷移機械

4.1　システムのモデル

本研究ではセキュリティプロトコルのモデル化に観測遷移機械 (Observational Transition System) を用いる [3, 4, 5]. 観測遷移機械では対象システムに関連する値がシステムの実行に伴う状態遷移により, どのように変化するのかを状態の外部から観測することでシステムのモデルを作成する.

モデル化の対象となるシステムの状態空間を包含する状態空間 Υ の存在を仮定する. 状態空間の各要素は状態と呼ぶ. 観測遷移機械 S は, 観測の集合 O, 初期状態の集合 I, および条件付遷移規則の集合 Γ の三つにより定義される.

観測遷移機械 S の実行は初期状態にはじまり, 永続的に遷移規則の一つを選択し適用することによって得られる状態の無限列である. 状態が S の実行に現れるとき, その状態は到達可能であるという.

また, 観測遷移機械 S がある性質 p [1]を有するというのは, S の任意の到達可能状態において性質 p が成り立つことである. その検証は遷移規則の適用回数に関する帰納法を用いて行う. S が性質 p を有することの検証の概略は以下のとおりである.

- 基底
  任意の初期状態 v において $p(v)$ は真であることを示す.
- 帰納段階
  $p(v)$ が真である任意の状態 v において, 任意の遷移規則 $\tau \in \Gamma$ に対し, $p(\tau(v))$ が真であることを示す. ただし, 帰納段階ではしばしば補題を必要とする. 使用する補題を q とすると, $p(\tau(v))$ を示す代わりに $q(v) => p(\tau(v))$ を示せばよい.

4.2　CafeOBJ による観測遷移機械の記述

状態空間 Υ は H を隠蔽ソートとしたとき, `*[ H ]*`のように宣言することで表現する. 可視ソート $V_k(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$ および V は各データ型 $D_k(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$ および D を始代数に基づいて定義することで表現する.

[1]本研究で扱う性質は不変表明のみである.

観測遷移機械 S の観測 $o_{i_1,\ldots,i_m} \in O$ は CafeOBJ の観測演算で表現する. 観測 $o_{i_1,\ldots,i_m}$ に対応する観測演算は次のように宣言する.

$$\texttt{bop o :}\ H\ V_{i_1} \ldots V_{i_m}\ \texttt{->}\ V$$

初期条件 I(初期状態での各観測値) は初期状態を表す定数を宣言し, 各観測値を等式で定義する. 初期状態を表す定数を init とし, 次のように宣言する.

```
op init : -> H
```

ソート V_k の CafeOBJ 変数を $X_k(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$ とする. 観測 $o_{i_1,\ldots,i_m} \in O$ の初期値が $f(i_1, \ldots, i_m)$ で表される関数で与えられると,

$$\texttt{eq o(init,}X_{i_1}, \ldots, X_{i_m}) \ =\ f(i_1, \ldots, i_m)$$

と記述できる.

観測遷移機械 S の遷移規則 $\tau_{j_1,\ldots,j_n} \in \Gamma$ は CafeOBJ の作用演算で表現する. 遷移規則 $\tau_{j_1,\ldots,j_n}$ に対応する作用演算は次のように宣言する.

$$\texttt{bop a :}\ H\ V_{j_1} \ldots V_{j_n}\ \texttt{->}\ H$$

状態 W をソート H の CafeOBJ 変数とする. 状態 W に対応する遷移規則を適用した後の事後状態を $a(W, X_{j_1}, \ldots, X_{j_n})$ と表す.
$e - a(W, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})$ を遷移規則が効力のある状態で実行された場合の観測 O_i の事後状態における観測値に対応する CafeOBJ の項とし, $c - a(W, X_{j_1}, \ldots, X_{j_n})$ を遷移規則の効力条件 $c_{\tau_{j_1,\ldots,j_n}}$ に対応する CafeOBJ の項とする.

効力条件が真である状態における遷移規則 $\tau_{j_1,\ldots,j_n} \in \Gamma$ の実行に伴う, 観測 $o_{i_1,\ldots,i_m} \in O$ の観測値の変化は次のように記述する.

$$\texttt{ceq o}(a(W, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m}) \ =\ \ e - a(W, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m})$$
$$\texttt{if}\ \ \ c - a(W, X_{j_1}, \ldots, X_{j_n})$$

効力条件が偽であればどの観測値も変化しないので次のように記述する.

$$\texttt{ceq}\ a(W, X_{j_1}, \ldots, X_{j_n}) \ =\ W\ \texttt{if not}\ c - a(W, X_{j_1}, \ldots, X_{j_n})$$

効力条件の真偽に関わらず観測値を変化させない場合は

$$\texttt{ceq o}(a(W, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m}) \ =\ \texttt{o}(W, X_{i_1}, \ldots, X_{i_m})$$

のように記述する.

4.3 Coqによる観測遷移機械の記述

状態空間 Υ はその集合を表すデータ型 H として帰納的に定義することで表現する. 初期状態はデータ型 H を構成する定数initとして表現する. 各データ型 $D_k(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$ および D も同じくその集合を帰納的に定義することによって表現する.

観測遷移機械Sの遷移規則 $\tau_{j_1,\ldots,j_n} \in \Gamma$ はデータ型 H のコンストラクタとして表現する. つまり, データ型 H には遷移規則 $\tau_{j_1,\ldots,j_n}$ に対応するコンストラクタが存在し, その型は

$$H \texttt{ -> } D_{j_1} \texttt{ -> } \ldots \texttt{ -> } D_{j_n} \texttt{ -> } H$$

となっている.

観測遷移機械Sの観測 $o_{i_1,\ldots,i_m} \in O$ はそれぞれ一つの述語として帰納的に定義することで表現する. つまり, 型が

$$H \texttt{ -> } D_{i_1} \texttt{ -> } \ldots \texttt{ -> } D_{i_m} \texttt{ -> } D \texttt{ -> Prop}$$

である帰納的述語を定義することによって観測 $o_{i_1,\ldots,i_m} \in O$ に対応するものを表現する.

初期条件 I はこの帰納的述語の定義の一部としてスキーマとその証明の名前の組で以下のように記述し, 定義する. ここで X_k は型が D_k である変数であり, $f(i_1, \ldots, i_m)$ は観測の初期値を表す関数とする.

$$\texttt{o\_init} : (X_{i_1} : D_{i_1} \ldots, X_{i_m} : D_{i_m})(\texttt{o init } X_{i_1} \ldots X_{i_m}\ f(i_1, \ldots, i_m))$$

遷移規則による観測値の変化もスキーマとその証明の名前の組で以下のように記述し, 定義する. ここで W は型が H である変数であり, $a(W, X_{j_1}, \ldots, X_{j_n})$ は状態 W に対応する遷移規則を適用した後の事後状態を表す. また, $e - a(W, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})$ は遷移規則が効力のある状態で実行された場合の観測値を表し, $c - a(W, X_{j_1}, \ldots, X_{j_n})$ は遷移規則の効力条件 $c_{\tau_{j_1,\ldots,j_n}}$ を表す.

- 効力条件が真である場合

  $$\begin{aligned}\texttt{o\_aT} : {} & (W : H; N : D; X_{i_1} : D_{i_1} \ldots, X_{i_m} : D_{i_m};\ X_{j_1} : D_{j_1} \ldots, X_{j_m} : D_{j_m}) \\ & (\texttt{o } W\ N) \texttt{ /\textbackslash } \quad c - a(W, X_{j_1}, \ldots, X_{j_n}) \texttt{ ->} \\ & (\texttt{o } a(W, X_{j_1}, \ldots, X_{j_n})\ e - a(W, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m}))\end{aligned}$$

  $* \ e - a(W, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})$ は N を用いて表される.

- 効力条件が偽である場合

  $$\begin{aligned}\texttt{o\_aF} : {} & (W : H; N : D; X_{i_1} : D_{i_1} \ldots, X_{i_m} : D_{i_m}) \\ & (\texttt{o } W\ N) \texttt{ /\textbackslash } \quad \texttt{~}c - a(W, X_{j_1}, \ldots, X_{j_n}) \texttt{ -> } (\texttt{o } a(W, X_{j_1}, \ldots, X_{j_n})\ N)\end{aligned}$$

- 観測値を変化させない場合

  $$\begin{aligned}\texttt{o\_a} : {} & (W : H; N : D; X_{i_1} : D_{i_1} \ldots, X_{i_m} : D_{i_m};\ X_{j_1} : D_{j_1} \ldots, X_{j_m} : D_{j_m}) \\ & (\texttt{o } W\ N) \texttt{ -> } (\texttt{o } a(W, X_{j_1}, \ldots, X_{j_n})\ N)\end{aligned}$$

4.4 観測遷移機械の記述例

観測遷移機械の記述例として bank account 抽象機械の記述を行う. CafeOBJ 仕様および Coq 仕様を以下に示す.

・(CafeOBJ)

Nat は自然数を表すソートであり, <=,<,+,-などの演算とともに輸入されるモジュール INT で既に定義されている. Account は状態空間を表す隠蔽ソートである. init は初期状態を表す. balance は口座の残高を表す観測演算で, deposit および withdraw は口座に対する預金および引き出しを表す作用演算である. A は状態を, N は自然数を表すとすると, balance(A) は状態 A における残高を, deposit(A,N) は状態 A で金額 N を預金した後の状態, withdraw(A,N) は状態 A で金額 N の引き出しを試みた後の状態を表す. 初期状態では残高は 0 で, 等式で規定する. deposit および withdraw の適用により残高がどのように変化するかを等式で規定する.

```
mod* ACCOUNT {
  pr(INT)  *[Account]*
  op init : -> Account
  bop balance : Account -> Nat
  bops deposit withdraw : Account Nat -> Account
  var N : Nat  var A : Account
  eq balance(init) = 0 .
  eq balance(deposit(A,N)) = balance(A) + N .
  ceq balance(withdraw(A,N)) = balance(A) - N if N <= balance(A) .
  ceq balance(withdraw(A,N)) = balance(A) if balance(A) < N .
}
```

・(Coq)

自然数に関する演算 le,lt,plus,minus [2]は既に通常の意味で定義されているものとする. 状態空間 Account を帰納的に定義する. Account の要素を状態と呼ぶことにする. init は初期状態を表す. a は状態を, n は自然数を表すとすると, (deposit a n) は状態 a で金額 n を預金した後の状態, (withdraw a n) は状態 a で金額 n の引き出しを試みた後の状態を表す. 口座の残高は, 状態と自然数を引数にとる述語 balance で表す. (balance a n) は状態 a における残高が金額 n であることを表している.

[2]それぞれ CafeOBJ 仕様における<=,<,+,-に対応

```
Inductive Account : Set :=
  init : Account
| deposit : Account -> nat -> Account
| withdraw : Account -> nat -> Account.

Inductive balance : Account -> nat -> Prop :=
  bal_init : (balance init O)
| bal_depo : (a:Account;n,m:nat)
(balance a n) -> (balance (deposit a m) (plus n m))
| bal_withT : (a:Account;n,m:nat)
(balance a n) /\ (le m n) -> (balance (withdraw a m) (minus n m))
| bal_withF : (a:Account;n,m:nat)
(balance a n) /\ (lt n m) -> (balance (withdraw a m) n).
```

第5章　STSプロトコル

5.1　Diffie-Hellman鍵交換プロトコル

セキュリティプロトコルの代表的な例としてDiffie-Hellman鍵交換プロトコル[1][2]がある. D.H.鍵交換プロトコルは1976年にDiffieとHellmanにより提案されたプロトコルで, 共通鍵暗号を作成する鍵を交換する.

共通鍵暗号は暗号化鍵と復号化鍵が同一である暗号で, 暗号化して通信を行う二者が同じ鍵を共有している. 主体Xと主体Yとの間の共有鍵をK_{XY}とし, これにより暗号化した文を$\{\cdots\}_{K_{XY}}$と書く. 暗号文は共通鍵の持ち主である主体X,Yにしか復号することはできない.

D.H.鍵交換プロトコルは離散対数問題の計算不可能性を利用し, 事前の秘密共有なしに共通鍵を交換する. 具体的には, 共通のDiffie-Hellmanパラメータ[2] pを利用して作成したお互いの公開値EX_X,EX_Yの交換を行うことにより実現する. 主体Xが発生した乱数をxとすると, 主体Xの公開値EX_Aは$g^x \ mod \ q$である. 離散対数問題の計算不可能性とは, この$g^x \ mod \ q$およびpからxを求めることはできないという性質のことである. よって, 主体が発生する乱数はその主体のみが所有する秘密の値になる. また, 主体Yの乱数をyとすると, 共通鍵K_{XY}は$g^{xy} \ mod \ q$である. この共通鍵を主体Xは主体Yの公開値EX_Yのx乗を, 主体Yは主体Xの公開値EX_Xのy乗を計算することによって得ることができる. ネットワーク上を流れる情報p, EX_A, EX_Bからx, yを求めることはできないので, x, yの所有者である主体X,Yのみが計算できる共通鍵$g^{xy} \ mod \ q$を交換できる.

しかしながらこのプロトコルには, なりすましやメッセージの横取り等の不正を働く侵入者のman-in-the-middle攻撃によって, 誤った鍵交換を行ってしまうという欠陥が存在する. 欠陥の詳細は以下の通りである.

Message1.　$A \rightarrow B:$　p, EX_A
Message2.　$B \rightarrow A:$　EX_B,$\{EX_B\}_{K_{AB}}$

Diffie-Hellman鍵交換プロトコル

[1]以後D.H.鍵交換プロトコルと呼ぶ.
[2]十分大きな素数qとそれ以下の整数gの組. 以後D.H.パラメータと呼ぶ.

このプロトコルにおいて, 主体 C が不正をはたらく.

- STEP1　$A \to C : p, EX_A$
  主体 C は, 主体 B と通信しようとする主体 A の Message1 を横取りする.
- STEP2　$C \to B : p, EX_C$
  主体 C は主体 B に, Message1 を主体 A として送る. 主体 B は, 主体 C との共通鍵を得ることになる.
- STEP3　$B \to C : EX_B, \{EX_B\}_{K_{BC}}$
  主体 C は, 主体 A と通信しようとする主体 B の Message2 を横取りする.
- STEP4　$C \to A : EX_C, \{EX_C\}_{K_{AC}}$
  主体 C は主体 A に, Message2 を主体 B として送る. 主体 A は, 主体 C との共通鍵を得ることになる.

得られた共通鍵により暗号化された主体 A および主体 B の暗号文はすべて主体 C により解読されてしまうことになる.

また, 計算量クラスの観点から言うと NP 完全でも P でもないクラスに属する離散対数問題に対する多項式時間アルゴリズムも存在する [1994 Shor]. しかし本論文ではその研究の性格上, 離散対数問題は計算不可能であるとする.

5.2　STS プロトコル

STS(station-to-station) プロトコルは, 1992 年に W. Diffie, P. van Oorschot, M. Wiener らによって共通鍵暗号方式を用いた認証鍵交換プロトコルとして提案されたプロトコル [1] で, D.H. 鍵交換プロトコルに存在する脆弱性への対抗を主目的としている. 具体的には, 通信を行う双方がデジタル署名と公開鍵証明書を使用することによりお互いを認証することで目的を実現している.

デジタル署名は, 送信するメッセージに対し自分の秘密鍵で暗号化 (署名) を行うことで, 送信者 (署名) が本物であることの確認を送信者の公開鍵を手に入れることのできる者が行えることを可能にするものである. 主体 X の秘密鍵を s_X と書き, これにより署名されたメッセージ Msg を $\{Msg\}_{s_X}$ と書く. 通常デジタル署名は短いメッセージに対して行うことが効果的であるため, 一方向性ハッシュ関数を適用したものに署名を行うが, 本論文では簡単のためこれを省略する.

公開鍵証明書は, 主体の名前と公開鍵を含んだ文に信頼できる認証局のデジタル署名がなされたものである. 信頼できる認証局は各主体の名前と公開鍵の組を間違いなく認識できる. そのため, 誤った名前と公開鍵の組を含んだ文に署名し, 誤った公開鍵証明書を発行することはない. 主体 X の名前と公開鍵を含んだ公開鍵証明書を $Cert_X$ と書く.

通信を行う各主体は, 共通に信頼できる認証局 T から自分の名前と公開鍵を含んだ証明書を受け取れる, かつ認証局 T の公開鍵を所持していると仮定すると, STS プロトコルの一つのセッションは次の 3 ステップで記述できる.

$$\begin{aligned}
&\text{Message1. } A \to B: \ p, EX_A \\
&\text{Message2. } B \to A: \ EX_B, Cert_B, \{\{EX_A, EX_B\}_{s_B}\}_{K_{AB}} \\
&\text{Message3. } A \to B: \ Cert_A, \{\{EX_A, EX_B\}_{s_A}\}_{K_{AB}}
\end{aligned}$$

STS プロトコルで使用する公開鍵証明書は各主体の名前と公開鍵の他に, 通信で使用する D.H. パラメータを一つ含んでいる. 主体 X の公開鍵を p_X と書けば, 主体 X に関する公開鍵証明書 $Cert_X$ は

$$Cert_X = \{X, p_X, p\}_{s_T}$$

である. ここでも簡単のため, 署名に伴うハッシュ関数は省略することにする.

次に STS プロトコルの手順を具体的に追う.

- STEP1　Message1 $A \to B: \ p, EX_A$

  主体 B と秘密の通信を行いたい主体 A は秘密の通信を行うために必要な共通鍵を交換するために, D.H. パラメータ p と乱数を生成し, 自分の公開値 EX_A を計算する. それを D.H. パラメータ p とともに主体 B に送信する.

- STEP2　Message2 $B \to A: EX_B, Cert_B, \{\{EX_A, EX_B\}_{s_B}\}_{K_{AB}}$

  Message1 を受信した主体 B は乱数を生成し, 送られてきた D.H. パラメータ p を使って自分の公開値 EX_B を計算する. また, 生成した乱数と送られてきた主体 A の公開値 EX_A を使って共通鍵 K_{AB} を計算し, 2 つの公開値に自身のデジタル署名を行った $\{EX_A, EX_B\}_{s_B}$ を暗号化する. その暗号文 $\{\{EX_A, EX_B\}_{s_B}\}_{K_{AB}}$ と自身の公開値 EX_B および公開鍵証明書 $Cert_B$ を主体 A に送信する.

- STEP3　Message3 $A \to B: Cert_A, \{\{EX_A, EX_B\}_{s_A}\}_{K_{AB}}$

  Message2 を受信した主体 A は送られてきた証明書 $Cert_B$ が認証局 T が発行したものであることをまず確認し, その上で送信者 B とパ D.H. ラメータ p が自身が送信した Message1 の送信相手と D.H. パラメータに一致することを確認する. 次に, 送られてきた主体 B の公開値 EX_B を使って共通鍵 K_{AB} を計算し, 暗号文 $\{\{EX_A, EX_B\}_{s_B}\}_{K_{AB}}$ を復号する. さらに電子署名 $\{EX_A, EX_B\}_{s_B}$ を検証し, 主体 B の署名がなされた公開値の組が Message1 で自分が送信した EX_A と Message2 で送られてきた EX_B の組であることを確認する. その後, 2 つの公開値に自身のデジタル署名を行った $\{EX_A, EX_B\}_{s_A}$ を共通鍵 K_{AB} で暗号化し, その暗号文 $\{\{EX_A, EX_B\}_{s_A}\}_{K_{AB}}$ と公開鍵証明書 $Cert_A$ を主体 B に送信する.

交換した鍵を用いた実際の通信を Message4,Message5 とする. Message2 を受け取った後, 主体 A は主体 B に対して自分のあるいは主体 B の秘密情報を含んだ暗号通信文を送

信する.

$$\text{Message4.}\quad A \to B:\ \{secret\}_{K_{AB}}$$

また, Message3 を受け取った主体 B は STEP3 の主体 A 同様, 送られてきた証明書, 暗号文内の署名, 署名された公開値の検証を行った後, 主体 A に対して自分のあるいは主体 A の秘密情報を含んだ暗号通信文を送信する.

$$\text{Message5.}\quad B \to A:\ \{secret\}_{K_{AB}}$$

タイミングは異なるものの, 交換した共通鍵を使って互いに秘密通信を行うことができる.

5.3 STS プロトコルの特徴

STS プロトコルには以下のような特徴があるとされている.

- Perfect Forward Security
  セッション毎に D.H. パラメータや乱数を変更する必要なしに, 安全性が保証される. 送信した情報の再利用によって, 安全性が脅かされることはない.
- Direct Authentication
  共通鍵の交換が行われたことが, プロトコルの実行のみで保証される. そのため, 共通鍵を暗号化して送るといった余分な動作が必要ない.

これらの特徴を有することは安全性と信頼性の検証を行うことによって示すことができる.

第6章　STSプロトコルの形式化

6.1　セキュリティプロトコルの仮定

システムを形式化しシュミレーションを行う際には, そのシステムがどのような状況でどのようなことが出来るのか, どのような性質を持っているのか, のようにあらかじめ仮定をしておく必要がある. またできる限り検証を簡潔に行えるようにするため, モデル化の部分で工夫を行ったものも仮定として含める. 本研究ではSTSプロトコルをモデル化するにあたって以下を仮定する.

- プロトコルの参加者に関する仮定
  STSプロトコルの参加者[1]として, 信頼できる主体・信頼できない主体の二種類を仮定する. 信頼できる主体とは, プロトコルに則った動作のみを行うプロトコルの正式な参加者である. 信頼できない主体とは, プロトコルに則った動作に加えて通信網を流れるメッセージを盗聴し, 手に入れた情報を基にメッセージの偽造を行う. 以後, 信頼できる主体を正規の参加者と, 信頼できない主体を侵入者として取り扱う. 侵入者についての仮定の詳細は後で記述する.

- 暗号の安全性に関する仮定
  本研究ではプロトコル自体に欠陥がないことを検証するので, 利用する暗号は完全なものであり, 復号鍵なしでは解読できないものとする.

- 通信網の安全性に関する仮定
  メッセージを媒介する通信網は安全なものではない. 一度ネットワークに入れられたメッセージは第三者によって盗聴することが可能である.

- D.H.パラメータの仮定
  十分大きな素数とそれ以下の整数の組という形の上では二つの値を持つものであるが, 検証に影響がないため一つのものとして扱う. またSTSプロトコルのPerfect Forward Securityの性質を考慮し, 再使用可能な値であると仮定した. 任意の主体が任意の値を使用できる.

[1]以後, 単に主体と呼ぶ.

- 乱数の仮定
  発生させた本人以外にその乱数は使われることはないものとし, 主体独自の値であるとする.

- 電子署名の仮定
  署名を行えるのは本人のみであるとする. 以後, ある媒体に署名を施したとき, そのもの自体を署名と呼ぶ.

- 署名の中身に関する仮定
  簡単のため, 署名を行い暗号化する公開値は Message2,Message3 ともに自分のものだけとする. このことにより暗号のアルゴリズムに関する攻撃が存在してしまう [1] が, プロトコルの動作を考える上では問題ない.

- 公開鍵証明書の仮定
  すべての主体は信頼できる認証局から自分の公開鍵証明書を受け取ることが出来るとする. この公開鍵証明書において, 主体とその公開鍵の組は常に正しいとする.

- セッション鍵の仮定
  STS プロトコルで交換する鍵は共通鍵であるので, 値としては一つのものである. しかし理解を容易にするため, 鍵は形の上で二つあり, 主体はもう一方の主体向けの鍵で暗号化を行うものとする. この鍵をセッション鍵と呼ぶ.

- 侵入者の仮定
  侵入者はプロトコルに則った動作を行う一方, 通信網を流れるメッセージの盗聴等の不正な動作を行い, プロトコルを攻撃する. 侵入者の行うことができる不正な動作は以下の通りである.

  - Messag1-3 について
    * 通信網を流れるメッセージを盗聴する.
    * 盗聴したメッセージに含まれるデータを収集する[2].
    * 収集したデータを基にメッセージを偽造し送信する.
  - Message4,5 について
    * 通信網を流れるメッセージを盗聴する.

[2]暗号文については侵入者向けのセッション鍵で暗号化されている場合のみ復号可能で，中の署名を収集することができる.

簡単のため, プロトコルの動作と直接関係ない通信である Message4,5 についてはメッセージの偽造の対象から外し, 盗聴のみを考えることとした. 以上より, 侵入者は通信網からのみ情報を収集することができ, 他の媒体から情報を収集することはできない. メッセージの偽造に関しては, 利用可能なものは何でも利用し何でも作ることができる.

6.2 モデル化で使用するデータ型の定義

STS プロトコルのモデル化で使用するデータ型を CafeOBJ と Coq それぞれで定義する. なお都合上, CafeOBJ 仕様の一部 (モジュールの公理部) は割愛しているものがある. 定義の全体については, 付録 A を参照して欲しい.

6.2.1 主体

・(CafeOBJ) モジュール PRINCIPAL は主体を定義する.

```
mod* PRINCIPAL {
  [Principal]
  op intruder : -> Principal
  op _=_ : Principal Principal -> Bool {comm}
  var P : Principal
  eq (P = P) = true .
}
```

Principal は主体のためのソートである. intruder は Principal の一部として, 汎用の侵入者をモデル化している. 緩いモジュールで定義を行っているため, ソートが Principal である定数をその都度追加で宣言することによって正規の参加者を表す. 以後, CafeOBJ での説明において p,p1,p2 はソートが Principal である定数を表すものとする.

Bool は論理式のためのソートである. 演算子_=_は, 定義する各データ型に対する等しさを判定する. comm は演算子_=_に与えられた属性で, 演算が交換律を満たすことを宣言する. 演算子_=_は公理部において等式を宣言することにより, 「同じ形をした項は等しい」という意味で定義する.

・(Coq) 主体の集合 Principal を帰納的に定義する.

```
Inductive Principal : Set :=
  intruder : Principal
| P : nat -> Principal.
```

主体は (P 0) や (P (s 0)) や intruder などの項によって表される. P は正規の参加者を表すデータ構成子で, 侵入者を表す構成子 intruder とは区別される. 引数の nat はその正規の参加者を区別するためのものである.

データ型の等しさの判定を行う演算は, 同じ形をした項を Coq は等しいと判定するため, この場合は含まなくても良い.

6.2.2 Diffie-Hellman パラメータ

・(CafeOBJ) モジュール PARAMETER は D.H. パラメータを定義する.

```
mod* PARAMETER {
  [Parameter]
  op _=_ : Parameter Parameter -> Parameter

  -- 公理部 略
}
```

Parameter は D.H. パラメータのためのソートである. 緩いモジュールで定義を行っているために正規の参加者と同様, D.H. パラメータはソートが Parameter である定数によって表される. 以後, CafeOBJ での説明において pa はソートが Parameter である定数を表すものとする.

・(Coq) D.H. パラメータの集合 Param を帰納的に定義する.

```
Inductive Param : Set := PA : nat -> Param .
```

D.H. パラメータは (PA 0) や (PA (s 0)) などの項によって表される. PA は D.H. パラメータのコンストラクタである. 引数の nat はパラメータを区別する.

6.2.3 乱数

・(CafeOBJ) モジュール RANDOM は乱数を定義する.

```
mod! RANDOM {
  pr(PRINCIPAL)
  [Random]
  op rand : Principal -> Random
  op pr : Random -> Principal
  op _=_ : Random Random -> Bool {comm}
```

```
  -- 公理部 略
}
```

乱数は rand(p) や rand(intruder) などの項によって表される. 演算子 rand は乱数のデータ構成子である. 引数の pr は乱数を発生した主体を表している. pr はメタな情報であり, 生成した主体および検証者以外これを利用することはできない. すなわち, 侵入者によってこの情報の偽造はできないことを表す.

・(Coq) 乱数の集合 Random を帰納的に定義する.

```
Inductive Random : Set := RAND : Principal -> Random.
```

乱数は (RAND (P O)) や (RAND intruder) などの項によって表される. RAND は乱数のデータ構成子であり, 引数の Principal は乱数の発生者を表す.

6.2.4 公開値

・(CafeOBJ) モジュール EXPONENT は公開値を定義する.

```
mod! EXPONENT {
  pr(PARAMETER + RANDOM)
  [Exponent]
  op expo : Parameter Random -> Exponent
  op para : Exponent -> Parameter
  op rand : Exponent -> Random
  op _=_ : Exponent Exponent -> Bool {comm}

  -- 公理部 略
}
```

公開値は expo(pa,rand(p)) や expo(pa,rand(intruder)) などの項によって表される. 演算子 expo は公開値のデータ構成子である. 引数の para,rand はそれぞれ, この公開値を計算するときに使用した D.H. パラメータおよび乱数を表している. para,rand はメタな情報である. 公開値の作成者は乱数の発生者に等しい.

・(Coq) 公開値の集合 Exponent を帰納的に定義する.

```
Inductive Exponent : Set := EXPO : Param -> Random -> Exponent.
```

公開値は (EXPO (PA 0) (RAND (P 0))) や (EXPO (PA 0) (RAND intruder)) などの項によって表される. データ構造は CafeOBJ のものと同じである. 便宜のため, 公開値の作成者を求める関数 whoz_expo をラムダ記法で定義する.

```
Definition whoz_expo [EX:Exponent] :=
  Cases EX of (EXPO _ (RAND p)) => p end.
```

6.2.5 セッション鍵

・(CafeOBJ) モジュール SKEY はセッション鍵を定義する.

```
mod! SKEY {
  pr(COMPONENT)
  [Skey]
  op skey : Component Random -> Skey
  op exp : Skey -> Exponent
  op rdm : Skey -> Random
  op _=_ : Skey Skey -> Bool {comm}
  op rsk : Skey -> Bool
  var SK : Skey
  eq rsk(SK) = (pr(rand(exp(SK))) = intruder) .

  -- その他の公理部 略
}
```

セッション鍵は skey(expo(pa,rand(p1)),rand(p2)) などの項によって表される. 演算子 skey はセッション鍵のデータ構成子である. 引数の exp,rdm はそれぞれ, このセッション鍵を計算するときに使用した離散対数および乱数を表している. para,rand はメタな情報である. セッション鍵の作成者は乱数の発生者に等しく, 離散対数の作成者向けの鍵である. 演算 rsk はセッション鍵が侵入者用のものか, つまりこの鍵で暗号化された暗号文を侵入者が複合可能かを判定する.

・(Coq) セッション鍵の集合 Skey を帰納的に定義する.

```
Inductive Skey : Set := SKEY : Exponent -> Random -> Skey.
```

セッション鍵は (SKEY (EXPO (PA 0) (RAND (P 0))) (RAND (P (s 0)))) などの項によって表される. データ構造は CafeOBJ のものと同じである. 便宜のため, セッション鍵の利用者を求める関数 key_for をラムダ記法で定義する.

```
Definition key_for [SK:Skey] :=
  Cases SK of (SKEY ex _) => (whoz_expo ex) end.
```

6.2.6 公開鍵証明書

・(CafeOBJ) モジュール CERTIFICATE は公開鍵証明書を定義する.

```
mod! CERTIFICATE {
  pr(PRINCIPAL + PARAMETER)
  [Certificate]
  op cert : Principal Parameter -> Certificate
  op pr : Certificate -> Principal
  op pa : Certificate -> Parameter
  op _=_ : Certificate Certificate -> Bool {comm}

  -- 公理部 略
}
```

公開鍵証明書は cert(p,pa) や cert(intruder,pa) などの項によって表される. 演算子 cert は公開鍵証明書のデータ構成子である. 引数の pr,pa はそれぞれ, この公開鍵証明書に記載されている公開鍵の持ち主および D.H. パラメータを表している.

・(Coq) 電子証明書の集合 Certificate を帰納的に定義する.

```
Inductive Certificate : Set :=
  CERT : Principal -> Param -> Certificate.
```

公開鍵証明書は (CERT (P O) (PA O)) や (CERT intruder (PA O)) などの項によって表される. データ構造は CafeOBJ のものと同じである.

6.2.7 電子署名

・(CafeOBJ) モジュール SIGN は電子署名を定義する.

```
mod! SIGN {
  pr(PRINCIPAL + EXPONENT)
  [Sign]
  op sign : Principal Exponent -> Sign
  op pr : Sign -> Principal
  op ex : Sign -> Exponent
  op _=_ : Sign Sign -> Bool {comm}

  -- 公理部 略
}
```

電子署名は sign(p,expo(pa,rand(p))) や sign(intruder,expo(pa,rand(intruder))) などの項によって表される. 演算子 sign は電子署名のデータ構成子である. 引数の pr は電子署名の署名者を, ex は署名される公開値を表している.

・(Coq) 電子署名の集合 Sign を帰納的に定義する.

```
Inductive Sign : Set := SIGN : Principal -> Exponent -> Sign.
```

電子署名は (SIGN (P O) (EXPO (PA O) (RAND (P O)))) などの項によって表される. データ構造は CafeOBJ のものと同じである.

6.2.8 暗号文

・(CafeOBJ) モジュール CIPHER は Message2 および Message3 に現れる暗号文を定義する.

```
mod! CIPHER {
  pr(SIGN + SKEY)
  [Cipher]
  op enc : Skey Sign -> Cipher
  op skey : Cipher -> Skey
  op sign : Cipher -> Sign
  op _=_ : Cipher Cipher -> Bool {comm}
  var SK : Skey
  var SI : Sign
  eq skey(enc(SK,SI)) = SK .

  -- その他の公理部 略
}
```

暗号文は enc(skey(expo(pa,rand(p1)),rand(p2)),sign(p2,expo(pa,rand(p2)))) などの項によって表される. 演算子 enc は暗号文のデータ構成子である. 引数の skey は暗号化するセッション鍵を, sign は暗号化される署名を表している.

・(Coq) Message2 および Message3 に現れる暗号文の集合 Cipher を帰納的に定義する.

```
Inductive Cipher : Set := ENC : Skey -> Sign -> Cipher.
```

暗号文は

```
(ENC (SKEY (EXPO (PA O) (RAND (P O))) (RAND (P (s O))))
     (SIGN (P (s O)) (EXPO (PA O) (RAND (P (s O))))))
```

などの項によって表される. データ構造はCafeOBJのものと同じである. 便宜のため, 暗号文を暗号化しているセッション鍵を求める関数skeyをラムダ記法で定義する.

```
Definition skey [C:Cipher] := Cases C of (ENC x _) => x end.
```

6.2.9 暗号通信文

・(CafeOBJ) モジュールCIPHERMはMessage4およびMessage5である暗号通信文を定義する.

```
mod! CIPHERM {
  pr(SKEY)
  [CipherM]
  op enc : Principal Skey -> CipherM
  op for : CipherM -> Principal
  op key : CipherM -> Skey
  op _=_ : CipherM CipherM -> Bool {comm}
  var P : Principal
  var SK : Skey
  eq for(enc(P,SK)) = P .
  eq key(enc(P,SK)) = SK .

-- その他の公理部 略
}
```

暗号通信文はenc(p1,skey(expo(pa,rand(p1)),rand(p2))などの項によって表される. 演算子encは暗号通信文のデータ構成子である. 引数のfor,keyはそれぞれ, この暗号通信文を送る主体および暗号化するセッション鍵を表している.

・(Coq) Message4およびMessage5である暗号文の集合CipherMを帰納的に定義する.

```
Inductive CipherM : Set := ENCM : Principal -> Skey -> CipherM.
```

暗号通信文は

```
(CipherM (P 0) (SKEY (EXPO (PA 0) (RAND (P 0))) (RAND (P (s 0)))))
```

などの項によって表される. データ構造はCafeOBJのものと同じである. 便宜のため, 誰に向けてのメッセージかを求める関数forと暗号文を暗号化しているセッション鍵を求める関数skey2をラムダ記法で定義する.

```
Definition for [CM:CipherM] := Cases CM of (ENCM x _) => x end.
Definition skey2 [CM:CipherM] := Cases CM of (ENCM _ y) => y end.
```

6.2.10 メッセージ

・(CafeOBJ) モジュール Msg はプロトコルに関連するメッセージ五種類を定義する.

```
mod! MSG {
  pr(CERTIFICATE + CIPHER + CIPHERM)
  [Msg]
  op m1 : Principal Principal Principal Parameter Exponent -> Msg
  op m2 : Principal Principal Exponent Certificate Cipher -> Msg
  op m3 : Principal Principal Certificate Cipher -> Msg
  op m4 : Principal CipherM -> Msg
  op m5 : Principal CipherM -> Msg
  --
  ops m1? m2? m3? m4? m5? : Msg -> Bool
  ops creator sender receiver : Msg -> Principal
  op pa : Msg -> Parameter
  op ex : Msg -> Exponent
  op ce : Msg -> Certificate
  op ci : Msg -> Cipher
  op cm : Msg -> CipherM
  op _=_ : Msg Msg -> Bool {comm}

  -- 公理部 略
}
```

Msg はメッセージのためのソートであり m1(p1,p1,p2,pa,expo(pa,rand(p1))) などの項のソートである. 演算子 m1,m2,m3,m4,m5 はそれぞれ, プロトコルの Message1 から Message5 に対応したデータ構成子である. 五つの構成子の最初の引数は各メッセージの作成者を表すメタな情報である. 二番目以降の引数は各メッセージの送信者や受信者, または各メッセージに含まれているデータを表している.

演算子 m1?,m2?,m3?,m4?,m5?は通信網に加えられたメッセージが Message1 から Message5 のどのメッセージなのかを判別するために用いる. 残りの演算子は射影関数である.

・(Coq) プロトコルに関連するメッセージの集合 Msg を帰納的に定義する.

```
Inductive Msg : Set :=
  M1 : Principal -> Principal -> Principal ->
       Param -> Component -> Msg
| M2 : Principal -> Principal -> Component ->
       Certificate -> Cipher -> Msg
| M3 : Principal -> Principal -> Certificate -> Cipher -> Msg
| M4 : Principal -> CipherM -> Msg
| M5 : Principal -> CipherM -> Msg.
```

メッセージは (M1 (P O) (P O) (P (s O)) (PA O) (EXPO (PA O) (RAND (P O)))) などの項によって表される. データ構造は CafeOBJ のものと同じである.

6.3 システムに関連する値の定義

STS プロトコルの状態空間に関連する値として, 通信網を介して送信済みであるメッセージの集合を表すデータ型ネットワーク[3] を定義する.

・(CafeOBJ) モジュール NETWORK はネットワークを定義する. またモジュールでは, 侵入者がネットワークから取得できる公開値, 電子署名, 暗号文および取得し解読できる暗号通信文を定義する. ネットワークはメッセージの多重集合として定義する. 送信されネットワークに入れられたメッセージはネットワークから取り除かれることはない. 従って, ネットワークが空である場合はネットワークに一度もメッセージが送信されていないことを表している. ここではパラメタ付きモジュール BAG と COLLECTION を次のように具象化し, モジュール内のソートの名前を変更する. 記号->の左辺に現れるソート名を右辺の名前に変更する.

モジュール BAG では, 汎用の多重集合のためのソート Bag が, 空の多重集合を表す定数 void と多重集合の構成子を表す', ', および要素が多重集合に含まれているか判定する\in の二つの演算子とともに宣言されている. また, モジュール COLLECTION では, 汎用のものの集まりのためのソート Col が, 要素が多重集合に含まれているか判定する演算子\in とともに宣言されている. BAG および COLEECTION の記述については付録 A を参照して欲しい.

[3]以後単にネットワークというとき, データ型の名前もしくはデータ型が表す送信済みメッセージの集合のことを指すものとする.

```
pr(BAG(MSG)*{sort Bag -> Network})
pr(COLLECTION(EXPONENT)*{sort Col -> ColExpo})
pr(COLLECTION(SIGN)*{sort Col -> ColSign})
pr(COLLECTION(CIPHER)*{sort Col -> ColCipher})
pr(COLLECTION(CIPHERM)*{sort Col -> ColCipherM})
```

このモジュールの指標は次のようになる.

```
op cex : Network -> ColExpo
op csi : Network -> ColSign
op cci : Network -> ColCipher
op ccm : Network -> ColCipherM
```

これらの演算子は侵入者がネットワークから収集できる情報を表す演算子であり, モジュールの公理部において図 6.1 のように等式集合を宣言することによって定義する.

```
-- cex: 侵入者がネットワークから収集できる公開値を定義
eq EX \in cex(void) = false .
ceq EX \in cex(M,NW) = true
if m1?(M) and not(creator(M) = intruder) and ex(M) = EX .
ceq EX \in cex(M,NW) = true
if m2?(M) and not(creator(M) = intruder) and ex(M) = EX .
ceq EX \in cex(M,NW) = EX \in cex(NW)
if not(m1?(M) and not(creator(M) = intruder) and ex(M) = EX) and
   not(m2?(M) and not(creator(M) = intruder) and ex(M) = EX) .

-- csi: 侵入者がネットワークから収集できる電子署名を定義
eq SI \in csi(void) = false .
ceq SI \in csi(M,NW) = true
if m2?(M) and rsk(skey(ci(M))) and not(creator(M) = intruder) and
   sign(ci(M)) = SI .
ceq SI \in csi(M,NW) = true
if m3?(M) and rsk(skey(ci(M))) and not(creator(M) = intruder) and
   sign(ci(M)) = SI .
ceq SI \in csi(M,NW) = SI \in csi(NW)
if not(m2?(M) and rsk(skey(ci(M))) and
   not(creator(M) = intruder) and sign(ci(M)) = SI) and
   not(m3?(M) and rsk(skey(ci(M))) and
   not(creator(M) = intruder) and sign(ci(M)) = SI) .

-- cci: 侵入者がネットワークから収集できる暗号文を定義
eq CI \in cci(void) = false .
ceq CI \in cci(M,NW) = true
if m2?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI .
ceq CI \in cci(M,NW) = true
if m3?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI .
ceq CI \in cci(M,NW) = CI \in cci(NW)
if not(m2?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI) and
   not(m3?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI) .

-- ccm: 侵入者がネットワークから収集でき，かつ解読できる暗号通信文を定義
eq CM \in ccm(void) = false .
ceq CM \in ccm(M,NW) = true
if m4?(M) and rsk(key(cm(M))) and cm(M) = CM .
ceq CM \in ccm(M,NW) = true
if m5?(M) and rsk(key(cm(M))) and cm(M) = CM .
ceq CM \in ccm(M,NW) = CM \in ccm(NW)
if not(m4?(M) and rsk(key(cm(M))) and cm(M) = CM) and
   not(m5?(M) and rsk(key(cm(M))) and cm(M) = CM) .
```

図 6.1　　モジュール NETWORK の公理部

・(Coq) ネットワークはメッセージのリストとして定義する. またその他に, 侵入者がネットワークから取得できる公開値, 署名, 暗号文および取得し解読できる暗号通信文を定義する. それらもそれぞれの集合を要素として持つリストとして定義する.

```
Definition Network := (list Msg).
Definition ColExpo := (list Exponent).
Definition ColSign := (list Sign).
Definition ColCipher := (list Cipher).
Definition ColCipherM := (list CipherM).
```

Coq においても侵入者がネットワークから収集できる情報の定義を行う. Coq ではこれらを以下のような帰納的述語を定義することによって行う.

(cex ex n) は, 侵入者がネットワーク n から公開値 ex を取得できることを表す帰納的述語である (図 6.2).

(csi si n) は, 侵入者がネットワーク n から電子署名 si を取得できることを表す帰納的述語である (図 6.2).

(cci c n) は, 侵入者がネットワーク n から暗号文 c を取得できることを表す帰納的述語である (図 6.3).

(ccm cm n) は, 侵入者がネットワーク n から暗号通信文 cm を取得でき, かつ解読できることを表す帰納的述語である (図 6.3).

```
Inductive cex : Exponent -> Network -> Prop :=
  cex1 : (ex:Exponent;n:Network;m:nat;p1,p2:Principal;pa:Param)
          (cex ce (cons (M1 (P m) p1 p2 pa ex) n))
| cex2 : (ex1,ex2:Exponent;n:Network;p1,p2,p3:Principal;pa:Param)
          (cex ex1 n) -> ~ex1=ex2
       -> (cex ex1 (cons (M1 p1 p2 p3 pa ex2) n))
| cex3 : (ex:Exponent;n:Network;m:nat;p:Principal;ce:Certificate;c:Cipher)
          (cex ex (cons (M2 (P m) p ex ce c) n))
| cex4 : (ex1,ex2:Exponent;n:Network;p1,p2:Principal;ce:Certificate;c:Cipher)
          (cex ex1 n) -> ~ex1=ex2
       -> (cex ex1 (cons (M2 p1 p2 ex2 ce c) n))
| cex5 : (ex:Exponent;n:Network;p1,p2:Principal;ce:Certificate;c:Cipher)
          (cex ex n)
       -> (cex ex (cons (M3 p1 p2 ce c) n))
| cex6 : (ex:Exponent;n:Network;p:Principal;cm:CipherM)
          (cex ex n)
       -> (cex ex (cons (M4 p cm) n))
| cex7 : (ex:Exponent;n:Network;p:Principal;cm:CipherM)
          (cex ex n)
       -> (cex ex (cons (M5 p cm) n)).

Inductive csi : Sign -> Network -> Prop :=
  csi1 : (si:Sign;n:Network;m:nat;p1,p2,p3:Principal;pa:Param;ex:Exponent)
          (csi si n)
       -> (csi si (cons (M1 p1 p2 p2 pa ex) n))
| csi2 : (si:Sign;n:Network;m:nat;p:Principal;ex:Exponent;ce:Certificate)
         (sk:Skey)
          (rsk sk)
       -> (csi si (cons (M2 (P m) p ex ce (ENC sk si)) n))
| csi3 : (si1,si2:Sign;n:Network;m:nat;p1,p2:Principal;ex:Exponent)
         (ce:Certificate;sk:Skey)
          (csi si1 n) -> ~si1=si2
       -> (csi si1 (cons (M2 p1 p2 ex ce (ENC sk si2)) n))
| csi4 : (si:Sign;n:Network;m:nat;p:Principal;ce:Certificate;sk:Skey)
          (rsk sk)
       -> (csi si (cons (M3 (P m) p ce (ENC sk si)) n))
| csi5 : (si1,si2:Sign;n:Network;m:nat;p1,p2:Principal;ce:Certificate)
         (sk:Skey)
          (csi si1 n) -> ~si1=si2
       -> (csi si1 (cons (M3 p1 p2 ce (ENC sk si2)) n))
| csi6 : (si:Sign;n:Network;p:Principal;cm:CipherM)
          (csi si n)
       -> (csi si (cons (M4 p cm) n))
| csi7 : (si:Sign;n:Network;p:Principal;cm:CipherM)
          (csi si n)
       -> (csi si (cons (M5 p cm) n)).
```

図 6.2　　侵入者がネットワークから収集できる情報の定義 (1/2)

```
Inductive cci : Cipher -> Network -> Prop :=
  cci1 : (c:Cipher;n:Network;m:nat;p1,p2,p3:Principal;pa:Param;ex:Exponent)
          (cci c n)
       -> (cci c (cons (M1 p1 p2 p2 pa co) n))
| cci2 : (c:Cipher;n:Network;m:nat;p:Principal;ex:Exponent;ce:Certificate)
          ~(rsk (skey c))
       -> (cci c (cons (M2 (P m) p ex ce c) n))
| cci3 : (c1,c2:Cipher;n:Network;m:nat;p1,p2:Principal;ex:Exponent)
         (ce:Certificate)
          (cci c1 n) -> ~c1=c2
       -> (cci c1 (cons (M2 p1 p2 ex ce c2) n))
| cci4 : (c:Cipher;n:Network;m:nat;p:Principal;ce:Certificate)
          ~(rsk (skey c))
       -> (cci c (cons (M3 (P m) p ce c) n))
| cci5 : (c1,c2:Cipher;n:Network;m:nat;p1,p2:Principal;ce:Certificate)
          (cci c1 n) -> ~c1=c2
       -> (cci c1 (cons (M3 p1 p2 ce c2) n))
| cci6 : (c:Cipher;n:Network;p:Principal;cm:CipherM)
          (cci c n)
       -> (cci c (cons (M4 p cm) n))
| cci7 : (c:Cipher;n:Network;p:Principal;cm:CipherM)
          (cci c n)
       -> (cci c (cons (M5 p cm) n)).

Inductive ccm : CipherM -> Network -> Prop :=
  ccm1 : (cm:CipherM;n:Network;m:nat;p1,p2,p3:Principal;pa:Param;ex:Exponent)
          (ccm cm n)
       -> (ccm cm (cons (M1 p1 p2 p2 pa ex) n))
| ccm2 : (cm:CipherM;n:Network;m:nat;p1,p2:Principal;ex:Exponent)
         (ce:Certificate;c:Cipher)
          (ccm cm n)
       -> (ccm cm (cons (M2 p1 p2 ex ce c) n))
| ccm3 : (cm:CipherM;n:Network;m:nat;p1,p2:Principal;ce:Certificate;c:Cipher)
          (ccm cm n)
       -> (ccm cm (cons (M3 p1 p2 ce c) n))
| ccm4 : (cm:CipherM;n:Network;p:Principal)
          (rsk (skey2 cm))
       -> (ccm cm (cons (M4 p cm) n))
| ccm5 : (cm1,cm2:CipherM;n:Network;p:Principal)
          (ccm cm1 n) -> ~cm1=cm2
       -> (ccm cm1 (cons (M4 p cm2) n))
| ccm6 : (cm:CipherM;n:Network;p:Principal)
          (rsk (skey2 cm))
       -> (ccm cm (cons (M5 p cm) n))
| ccm7 : (cm1,cm2:CipherM;n:Network;p:Principal)
          (ccm cm1 n) -> ~cm1=cm2
       -> (ccm cm1 (cons (M5 p cm2) n)).
```

図6.3　　侵入者がネットワークから収集できる情報の定義(2/2)

6.4 STS プロトコルのモデル

6.4.1 CafeOBJ による記述

モジュール STS ではモデル化する観測遷移機械を振舞仕様で記述し, 定義する.

```
mod* STS {
pr(NETWORK)
*[System]*
-- any initial state
op init : -> System
-- observation operations
bop nw : System -> Network
-- for any initial state
eq nw(init) = void .

-- CafeOBJ variables
vars P P1 P2 : Principal
vars M M1 M2 M3 : Msg
vars PA PA1 PA2 : Parameter
vars CO CO1 CO2 : Component
var CE : Certificate    var SI : Sign
var CI : Cipher         var S : System

-- sending message
bop mes1 : System Principal Principal Parameter -> System
bop mes2 : System Principal Principal Msg -> System
bop mes3 : System Principal Principal Msg Msg -> System
bop mes4 : System Principal Principal Msg Msg -> System
bop mes5 : System Principal Principal Msg Msg -> System
-- faking message
-- for message1
bop fkm11 : System Principal Principal Parameter Parameter -> System
bop fkm12 : System Principal Principal Parameter Component -> System
-- for message2
bop fkm211 : System Principal Parameter Certificate Component Sign -> System
bop fkm212 : System Principal Component Certificate Component Sign -> System
bop fkm221 : System Principal Parameter Certificate Cipher -> System
bop fkm222 : System Principal Component Certificate Cipher -> System
-- for message3
bop fkm31 : System Principal Certificate Component Sign -> System
bop fkm32 : System Principal Certificate Cipher -> System

-- その他の公理部 後述
}
```

隠蔽ソート System は状態空間 Υ に対応している. 定数 init はプロトコルによるメッセージ交換がまだ行われていない任意の初期状態を表している.

観測演算 nw はプロトコルの動作によって送信されたメッセージの集合 (前節のネットワーク) を観察する. 初期状態 init における nw の観測値は void で定義する.

作用演算 mes1,mes2,mes3,mes4,mes5 はプロトコルに則った Message1 から Message5 までを送信する遷移規則に対応する. 作用演算 fkm11,fkm12 は侵入者が Message1 を偽造する遷移規則に対応する. 作用演算 fkm211,fkm212,fkm221,fkm222 は侵入者が Message2 を偽造する遷移規則に対応する. 作用演算 fkm31,fkm32 は侵入者が Message3 を偽造する遷移規則に対応する. これらの作用演算は, 図 6.4-6.6 に示すモジュール STS の公理部において等式集合を宣言することによって定義する.

作用演算 mes1 の二番目の引数はメッセージの作成, 送信者を表しており, 三番目の引数はメッセージの受信者を表している. 四番目の引数は送信するパラメータおよび公開値を作成するときに用いるパラメータを表している. (図 6.4)

作用演算 mes2 の二番目の引数はメッセージの作成, 送信者を表しており, 三番目の引数はメッセージの受信者を表している. 四番目の引数は二番目の引数で表される主体が送信したメッセージを表している. (図 6.4)

作用演算 mes3 の二番目の引数はメッセージの作成, 送信者を表しており, 三番目の引数はメッセージの受信者を表している. 二番目の引数で表される主体が送信したメッセージおよび受け取ったメッセージを, それぞれ四番目, 五番目の引数が表している. (図 6.4)

作用演算 mes4,mes5 の二番目の引数はメッセージの作成, 送信者を表しており, 三番目の引数は誰に向けてのメッセージかを表している. 二番目の引数で表される主体が送信したメッセージおよび受け取ったメッセージを, それぞれ四番目, 五番目の引数が表している. (図 6.4 および図 6.5)

作用演算 fkm11 には効力条件はない. 事後状態では偽造した Message1 をネットワークに追加する. (図 6.5)

作用演算 fkm12 の効力条件は, メッセージを偽造するのに必要な公開値を侵入者が取得していることで, 事後状態では偽造した Message1 をネットワークに追加する.
(図 6.5)

作用演算 fkm211,fkm212 の効力条件は, メッセージを偽造するのに必要な公開値および署名を侵入者が取得していることで, 事後状態では偽造した Message2 をネットワークに追加する. (図 6.5)

作用演算 fkm221 の効力条件は, メッセージを偽造するのに必要な暗号文を侵入者が取得していることで, 事後状態では偽造した Message2 をネットワークに追加する. (図 6.6)

作用演算 fkm222 の効力条件は, メッセージを偽造するのに必要な公開値および暗号文を侵入者が取得していることで, 事後状態では偽造した Message2 をネットワークに追加する. (図 6.6)

作用演算 fkm31 の効力条件は, メッセージを偽造するのに必要な公開値および署名を侵入者が取得していることで, 事後状態では偽造した Message3 をネットワークに追加する.
(図 6.6)

作用演算 fkm32 の効力条件は, メッセージを偽造するのに必要な暗号文を侵入者が取得していることで, 事後状態では偽造した Message3 をネットワークに追加する. (図 6.6)

```
-- for mes1
eq nw(mes1(S,P1,P2,PA)) = m1(P1,P1,P2,PA,expo(PA,rand(P1))) , nw(S) .

-- for mes2
op c-mes2 : System Principal Principal Msg -> Bool
eq c-mes2(S,P2,P1,M) =
M \in nw(S) and m1?(M1) and receiver(M) = P2 and sender(M) = P1 .
--
ceq nw(mes2(S,P2,P1,M)) =
m2(P2,P1,expo(pa(M),rand(P2)),cert(P2,pa(M)),
   enc(skey(ex(M),rand(P2)),sign(P2,expo(pa(M),rand(P2))))) , nw(S)
if c-mes2(S,P2,P1,M) .
ceq mes2(S,P2,P1,M) = S if not c-mes2(S,P2,P1,M) .

-- for mes3
op c-mes3 : System Principal Principal Msg Msg -> Bool
eq c-mes3(S,P1,P2,M1,M2) = M1 \in nw(S) and m1?(M1) and
creator(M1) = P1 and receiver(M1) = P2 and
M2 \in nw(S) and m2?(M2) and
pr(ce(M2)) = P2 and
pa(ce(M2)) = pa(M1) and
ex(M1) = exp(skey(ci(M2))) and
pr(sign(ci(M2))) = P2 and
ex(sign(ci(M2))) = ex(M2) .
--
ceq nw(mes3(S,P1,P2,M1,M2)) =
m3(P1,P2,cert(P1,pa(M1)),
   enc(skey(ex(M2),rand(ex(M1))),sign(P1,ex(M1)))) , nw(S)
if c-mes3(S,P1,P2,M1,M2) .
ceq mes3(S,P1,P2,M1,M2) = S if not c-mes3(S,P1,P2,M1,M2) .

-- for mes4
op c-mes4 : System Principal Principal Msg Msg -> Bool
eq c-mes4(S,P1,P2,M1,M2) =
not(P1 = intruder) and c-mes3(S,P1,P2,M1,M2) .
--
ceq nw(mes4(S,P1,P2,M1,M2)) = m4(P1,enc(P2,skey(ex(M2),rand(ex(M1))))) ,
                              nw(S) if c-mes4(S,P1,P2,M1,M2) .
ceq mes4(S,P1,P2,M1,M2) = S if not c-mes4(S,P1,P2,M1,M2) .
```

図 6.4　STS プロトコルの正規の Message-4 を送信する定義式 1

```
-- for mes5
op c-mes5 : System Principal Principal Msg Msg -> Bool
eq c-mes5(S,P2,P1,M2,M3) = not(P2 = intruder) and
M2 \in nw(S) and m2?(M2) and
creator(M2) = P2 and receiver(M2) = P1 and
M3 \in nw(S) and m3?(M3) and
pr(ce(M3)) = P1 and
pa(ce(M3)) = pa(ce(M2)) and
co(M2) = com(skey(ci(M3))) and
pr(sign(ci(M3))) = P1 and
co(sign(ci(M3))) = exp(skey(ci(M2))) .
--
ceq nw(mes5(S,P2,P1,M2,M3)) =
m5(P2,enc(P1,skey(ci(M2)))) , nw(S) if c-mes5(S,P2,P1,M2,M3) .
ceq mes5(S,P2,P1,M2,M3) = S if not c-mes5(S,P2,P1,M2,M3) .

-- for fkm11
eq nw(fkm11(S,P1,P2,PA1,PA2)) =
m1(intruder,P1,P2,PA1,expo(PA2,rand(intruder))) , nw(S) .

-- for fkm12
op c-fkm12 : System Exponent -> Bool
eq c-fkm12(S,EX) = EX \in cex(nw(S)) .
--
ceq nw(fkm12(S,P1,P2,PA,EX)) = m1(intruder,P1,P2,PA,EX) , nw(S)
if c-fkm12(S,EX) .
ceq fkm12(S,P1,P2,PA,EX) = S if not c-fkm12(S,EX) .

-- for fkm211
op c-fkm211 : System Component Sign -> Bool
eq c-fkm211(S,EX,SI) = EX \in cco(nw(S)) and SI \in csi(nw(S)) .
--
ceq nw(fkm211(S,P,PA,CE,EX,SI)) =
m2(intruder,P,comp(PA,rand(intruder)),CE,
   enc(skey(EX,rand(intruder)),SI)) , nw(S) if c-fkm211(S,EX,SI) .
ceq fkm211(S,P,PA,CE,EX,SI) = S if not c-fkm211(S,EX,SI) .

-- for fkm212
op c-fkm212 : System Component Component Sign -> Bool
eq c-fkm212(S,EX1,EX2,SI) =
EX1 \in cex(nw(S)) and EX2 \in cex(nw(S)) and SI \in csi(nw(S)) .
--
ceq nw(fkm212(S,P,EX1,CE,EX2,SI)) =
m2(intruder,P,EX1,CE,
   enc(skey(EX2,rand(intruder)),SI)) , nw(S) if c-fkm212(S,EX1,EX2,SI) .
ceq fkm212(S,P,EX1,CE,EX2,SI) = S if not c-fkm212(S,EX1,EX2,SI) .
```

図 6.5　正規の Message5 の送信，および侵入者のメッセージを偽造する定義式

```
-- for fkm221
op c-fkm221 : System Cipher -> Bool
eq c-fkm221(S,CI) = CI \in cci(nw(S)) .
--
ceq nw(fkm221(S,P,PA,CE,CI)) =
m2(intruder,P,comp(PA,rand(intruder)),CE,CI) , nw(S) if c-fkm221(S,CI) .
ceq fkm221(S,P,PA,CE,CI) = S if not c-fkm221(S,CI) .

-- for fkm222
op c-fkm222 : System Exponent Cipher -> Bool
eq c-fkm222(S,EX,CI) = EX \in cex(nw(S)) and CI \in cci(nw(S)) .
--
ceq nw(fkm222(S,P,EX,CE,CI)) =
m2(intruder,P,EX,CE,CI) , nw(S) if c-fkm222(S,EX,CI) .
ceq fkm222(S,P,EX,CE,CI) = S if not c-fkm222(S,EX,CI) .

-- for fkm31
op c-fkm31 : System Exponent Sign -> Bool
eq c-fkm31(S,EX,SI) = EX \in cex(nw(S)) and SI \in csi(nw(S)) .
--
ceq nw(fkm31(S,P,CE,EX,SI)) =
m3(intruder,P,CE,
   enc(skey(EX,rand(intruder)),SI)) , nw(S) if c-fkm31(S,EX,SI) .
ceq fkm31(S,P,CE,EX,SI) = S if not c-fkm31(S,EX,SI) .

-- for fkm32
op c-fkm32 : System Cipher -> Bool
eq c-fkm32(S,CI) = CI \in cci(nw(S))
--
ceq nw(fkm32(S,P,CE,CI)) = m3(intruder,P,CE,CI) , nw(S) if c-fkm32(S,CI) .
ceq fkm32(S,P,CE,CI) = S if not c-fkm32(S,CI) .
```

図 6.6　侵入者がメッセージを偽造する定義式 (残り)

6.4.2 Coq による記述

帰納的に定義される集合 System は状態空間 Υ に対応している. 定数 init はプロトコルによるメッセージ交換がまだ行われていない任意の初期状態を表している.

```
Inductive System : Set :=
  init : System
| mes1 : System -> Principal -> Principal -> Param -> System
| mes2 : System -> Principal -> Principal -> Param -> Exponent -> System
| mes3 : System -> Principal -> Principal -> Param -> Exponent ->
         Exponent -> System
| mes4 : System -> nat -> Principal -> Param -> Exponent ->
         Exponent -> System
| mes5 : System -> nat -> Principal -> Param -> Exponent ->
         Exponent -> System
| fkm11 : System -> Principal -> Principal -> Param -> Param -> System
| fkm12 : System -> Principal -> Principal -> Param -> Exponent -> System
| fkm211 : System -> Principal -> Param -> Certificate -> Exponent ->
           Sign -> System
| fkm212 : System -> Principal -> Exponent -> Certificate -> Exponent ->
           Sign -> System
| fkm221 : System -> Principal -> Param -> Certificate -> Cipher -> System
| fkm222 : System -> Principal -> Exponent -> Certificate -> Cipher -> System
| fkm31 : System -> Principal -> Certificate -> Exponent -> Sign -> System
| fkm32 : System -> Principal -> Certificate -> Cipher -> System.
```

作用演算 mes1 から fkm32 までによって観測演算 nw の観測する値がどのように変化するのかを, 効力条件のラムダ記法での定義を行いながら (図 6.7), 帰納的述語 (nw s n) として定義する (図 6.8-6.10). 各作用演算の効力条件, 事後状態で追加するメッセージは CafeOBJ の節のものと同じである.

```
Definition cmes2
[n:Network;p1,p2:Principal;pa:Param;ex:Exponent] : Prop :=
(EX Q:Principal | (In (M1 Q p1 p2 pa ex) n)).

Definition cmes3
[n:Network;p1,p2:Principal;pa:Param;ex1,ex2:Exponent] : Prop :=
(In (M1 p1 p1 p2 pa ex1) n) /\
(EX Q:Principal | (EX R:Randum |
(In (M2 Q p1 ex2 (CERT p2 pa) (ENC (SKEY ex1 R) (SIGN p2 ex2))) n))).

Definition cmes4
[n:Network;m:nat;p:Principal;pa:Param;ex1,ex2:Exponent] : Prop :=
(In (M1 (P m) (P m) p pa ex1) n) /\
(EX Q:Principal | (EX R:Randum |
(In (M2 Q (P m) ex2 (CERT p pa) (ENC (SKEY ex1 R) (SIGN p ex2))) n))).

Definition cmes5
[n:Network;m:nat;p:Principal;pa:Param;ex1,ex2:Exponent] : Prop :=
(In (M2 (P m) p ex1 (CERT (P m) pa)
        (ENC (SKEY ex2 (RAND (P m))) (SIGN (P m) ex1))) n) /\
(EX Q:Principal | (EX R:Randum |
(In (M3 Q (P m) (CERT p pa) (ENC (SKEY ex1 R) (SIGN p ex2))) n))).

Definition cfkm12 [n:Network;ex:Exponent] : Prop := (cex ex n).

Definition cfkm211 [n:Network;ex:Exponent;si:Sign] : Prop :=
(cex ex n) /\ (csi si n).

Definition cfkm212
[n:Network;ex1,ex2:Exponent;si:Sign] : Prop :=
(cex ex1 n) /\ (cex ex2 n) /\ (csi si n).

Definition cfkm221 [n:Network;c:Cipher] : Prop := (cci c n).

Definition cfkm222 [n:Network;ex:Exponent;c:Cipher] : Prop :=
(cex ex n) /\ (cci c n).

Definition cfkm31 [n:Network;ex:Exponent;si:Sign] : Prop :=
(cex ex n) /\ (csi si n).

Definition cfkm32 [n:Network;c:Cipher] : Prop := (cci c n).
```

図 6.7　Coq でのメッセージを送信するための条件の定義

```
Inductive nw : System -> Network -> Prop :=
  nw_init : (nw init (nil Msg))
| nw_mes1 : (s:System;n:Network;p1,p2:Principal;pa:Param)
             (nw s n)
          -> (nw (mes1 s p1 p2 pa)
             (cons (M1 p1 p1 p2 pa (COMP pa (RAND p1))) n))
| nw_mes2a : (s:System;n:Network;p1,p2:Principal;pa:Param;ex:Exponent)
             (nw s n) -> (cmes2 n p1 p2 pa ex) ->
             (nw (mes2 s p1 p2 pa ex)
             (cons (M2 p2 p1 (COMP pa (RAND p2)) (CERT p2 pa)
                             (ENC (SKEY ex (RAND p2))
                             (SIGN p2 (COMP pa (RAND p2))))) n))
| nw_mes2b : (s:System;n:Network;p1,p2:Principal;pa:Param;ex:Exponent)
             (nw s n) -> ~(cmes2 n p1 p2 pa ex) ->
             (nw (mes2 s p1 p2 pa ex) n)
| nw_mes3a : (s:System;n:Network;p1,p2:Principal)
             (pa:Param;ex1,ex2:Exponent)
             (nw s n) -> (cmes3 n p1 p2 pa ex1 ex2) ->
             (nw (mes3 s p1 p2 pa ex1 ex2)
             (cons (M3 p1 p2 (CERT p1 pa)
             (ENC (SKEY ex2 (RAND p1)) (SIGN p1 ex1))) n))
| nw_mes3b : (s:System;n:Network;p1,p2:Principal)
             (pa:Param;ex1,ex2:Exponent)
             (nw s n) -> ~(cmes3 n p1 p2 pa ex1 ex2) ->
             (nw (mes3 s p1 p2 pa ex1 ex2) n)
| nw_mes4a : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;ex1,ex2:Exponent)
             (nw s n) -> (cmes4 n m p pa ex1 ex2) ->
             (nw (mes4 s m p pa ex1 ex2)
             (cons (M4 (P m) (ENCM p (SKEY ex2 (RAND (P m))))) n))
| nw_mes4b : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;ex1,ex2:Exponent)
             (nw s n) -> ~(cmes4 n m p pa ex1 ex2) ->
             (nw (mes4 s m p pa ex1 ex2) n)
| nw_mes5a : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;ex1,ex2:Exponent)
             (nw s n) -> (cmes5 n m p pa ex1 ex2) ->
             (nw (mes5 s m p pa ex1 ex2)
             (cons (M5 (P m) (ENCM p (SKEY ex2 (RAND (P m))))) n))
| nw_mes5b : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;ex1,ex2:Exponent)
             (nw s n) -> ~(cmes5 n m p pa ex1 ex2) ->
             (nw (mes5 s m p pa ex1 ex2) n)
```

図 6.8　帰納的述語 nw の定義 (1/3)

```
| nw_fkm11 : (s:System;n:Network;p1,p2:Principal;pa1,pa2:Param)
             (nw s n)
          -> (nw (fkm11 s p1 p2 pa1 pa2)
             (cons (M1 intruder p1 p2 pa1 (COMP pa2 (RAND intruder))) n))
| nw_fkm12a : (s:System;n:Network;p1,p2:Principal;pa:Param;ex:Exponent)
             (nw s n) -> (cfkm12 n ex) ->
             (nw (fkm12 s p1 p2 pa ex)
             (cons (M1 intruder p1 p2 pa ex) n))
| nw_fkm12b : (s:System;n:Network;p1,p2:Principal;pa:Param;ex:Exponent)
             (nw s n) -> ~(cfkm12 n ex) ->
             (nw (fkm12 s p1 p2 pa ex) n)
| nw_fkm211a : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (ex:Exponent;si:Sign)
             (nw s n) -> (cfkm211 n ex si) ->
             (nw (fkm211 s p pa ce ex si)
             (cons (M2 intruder p (COMP pa (RAND intruder)) ce
                                  (ENC (SKEY co (RAND intruder)) si)) n))
| nw_fkm211b : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (ex:Exponent;si:Sign)
             (nw s n) -> ~(cfkm211 n ex si) ->
             (nw (fkm211 s p pa ce ex si) n)
| nw_fkm212a : (s:System;n:Network;p:Principal;ce:Certificate)
               (ex1,ex2:Exponent;si:Sign)
             (nw s n) -> (cfkm212 n ex1 ex2 si) ->
             (nw (fkm212 s p ex1 ce ex2 si)
             (cons (M2 intruder p ex1 ce
                   (ENC (SKEY ex2 (RAND intruder)) si)) n))
| nw_fkm212b : (s:System;n:Network;p:Principal;ce:Certificate)
               (ex1,ex2:Exponent;si:Sign)
             (nw s n) -> ~(cfkm212 n ex1 ex2 si) ->
             (nw (fkm212 s p ex1 ce ex2 si) n)
| nw_fkm221a : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (c:Cipher)
             (nw s n) -> (cfkm221 n c) ->
             (nw (fkm221 s p pa ce c)
             (cons (M2 intruder p (COMP pa (RAND intruder)) ce c) n))
| nw_fkm221b : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (c:Cipher)
             (nw s n) -> ~(cfkm221 n c) ->
             (nw (fkm221 s p pa ce c) n)
```

図 6.9　帰納的述語 nw の定義 (2/3)

```
| nw_fkm222a : (s:System;n:Network;p:Principal;ex:Exponent)
               (ce:Certificate;c:Cipher)
             (nw s n) -> (cfkm222 n ex c) ->
             (nw (fkm222 s p ex ce c) (cons (M2 intruder p ex ce c) n))
| nw_fkm222b : (s:System;n:Network;p:Principal;ex:Exponent)
               (ce:Certificate;c:Cipher)
             (nw s n) -> ~(cfkm222 n ex c) -> (nw (fkm222 s p ex ce c) n)
| nw_fkm31a : (s:System;n:Network;p:Principal;ce:Certificate)
              (ex:Exponent;si:Sign)
             (nw s n) -> (cfkm31 n ex si) ->
             (nw (fkm31 s p ce ex si)
             (cons (M3 intruder p ce
                   (ENC (SKEY ex (RAND intruder)) si)) n))
| nw_fkm31b : (s:System;n:Network;p:Principal;ce:Certificate)
              (ex:Exponent;si:Sign)
             (nw s n) -> ~(cfkm31 n ex si) ->
             (nw (fkm31 s p ce ex si) n)
| nw_fkm32a : (s:System;n:Network;p:Principal;ce:Certificate)
              (c:Cipher)
             (nw s n) -> (cfkm32 n c) ->
             (nw (fkm32 s p ce c) (cons (M3 intruder p ce c) n))
| nw_fkm32b : (s:System;n:Network;p:Principal;ce:Certificate)
              (c:Cipher)
             (nw s n) -> ~(cfkm32 n c) -> (nw (fkm32 s p ce c) n).
```

図 6.10　帰納的述語 nw の定義 (3/3)

第7章　STSプロトコルの検証

7.1　検証する性質

次の二つの表明が成り立つことを検証する.

表明 1 侵入者が解読できる暗号通信文は侵入者に向けた通信のみである.

表明 2 正規の参加者がもう一方の正規の参加者に暗号通信文を送信するとき,
受信者はその暗号通信文を解読可能である.

表明 1 を検証することで, 侵入者が他人に向けた暗号通信文を解読できないことを保証する. また表明 2 を検証することで, プロトコルの正式な参加者は正しく認証鍵を交換することを保証する.

上記の二つの表明は CafeOBJ 仕様および Coq 仕様として, それぞれ以下のように記述することができる. 表明 2 については Message4,Message5 の二つの場合が存在するが, 同様なので Message5 の場合は省略する.

・(CafeOBJ) 次のような項として記述することができる.

表明 1 任意の到達可能な s:System および任意の cm:CipherM に対して、

```
cm \in ccm(nw(s)) implies (for(cm) = intruder) .
```

表明 2 任意の到達可能な s:System および任意の m:Msg に対して、

```
MS \in nw(S) and m4?(MS) and
not(creator(MS) = intruder) and
not(for(cm(MS)) = intruder) implies
for(cm(MS)) = pr(rand(com(key(cm(MS))))) .
```

・(Coq) 次のようなスキーマで記述することができる.

表明 1

```
(s:System)(n:Network;cm:CipherM)
 (nw s n) /\ (ccm cm n) -> (for cm) = intruder.
```

表明 2

```
(s:System)(n:Network;m,l:nat;sk:Skey)
 (nw s n) /\ (In (M4 (P m) (ENCM (P l) sk)) n)
 -> (P l) = (key_for sk).
```

7.2 補題

各々の表明の検証は帰納法の仮定が弱いため, 帰納段階での検証が単独ではできない. そのため次のような補題が必要となる.

補題 1 Message2 中の署名内の構成要素の作成者は署名者と同じ.

補題 2 Message3 中の署名内の構成要素の作成者は署名者と同じ.

これら補題の検証にはさらに次の補題が必要となる.

補題 3 作成者が侵入者ではない message1 中の構成要素の作成者は, メッセージ作成者と同じ.

補題 4 侵入者が取得する署名内の構成要素の作成者は署名者と同じ.

補題 5 侵入者が取得する暗号文内の署名内の構成要素の作成者は署名者と同じ.

CafeOBJ 仕様および Coq 仕様での表現はそれぞれ付録 B, 付録 D を参照して欲しい.

7.3 表明の検証

・(CafeOBJ)

表明の検証を行うためにまずモジュール INV を宣言する. モジュール INV には前述の表明 1,2 の述語を表す演算子 inv100, inv200, inv300 を宣言する. 同時にそれらを定義する等式を宣言する.

```
op inv100 : System CipherM -> Bool
op inv200 : System Msg -> Bool
op inv300 : System Msg -> Bool
eq inv100(S,CM) = CM \in ccm(nw(S)) implies (for(CM) = intruder) .
eq inv200(S,MS) = MS \in nw(S) and m4?(MS) and
                  not(creator(MS) = intruder) and
                  not(for(cm(MS)) = intruder) implies
                  for(cm(MS)) = pr(rand(com(key(cm(MS))))) .
eq inv300(S,MS) = MS \in nw(S) and m5?(MS) and
                  not(creator(MS) = intruder) and
                  not(for(cm(MS)) = intruder) implies
                  for(cm(MS)) = pr(rand(com(key(cm(MS))))) .
```

ここで S はソート System の CafeOBJ 変数であり, MS,CM はそれぞれソート Msg,CipherM の CafeOBJ 変数である.

次にモジュール ISTEP で帰納段階で示す述語を表す演算子 istep100,istep200, istep300 を宣言し, 同時にそれらを定義する等式を宣言する.

```
ops s s' : -> System
op istep100 : CipherM -> Bool
op istep200 : Msg -> Bool
op istep300 : Msg -> Bool
eq istep100(CM) = inv100(s,CM) implies inv100(s',CM) .
eq istep200(MS) = inv200(s,MS) implies inv200(s',MS) .
eq istep300(MS) = inv300(s,MS) implies inv300(s',MS) .
```

ここで s は任意の状態を, s' は s で表される状態から遷移規則が適応された状態を表す.

本論文では表明 1, つまり inv100 で表される述語の証明について考え, 帰納法を用いた検証を記述した証明譜を作成する. まず, 任意の初期状態において成り立つことを示す. 証明譜の記述は以下のように行う.

```
open INV
red inv100(init,cm) .
close
```

CafeOBJ コマンド open で指定されたモジュールに宣言されているソート, 演算子および等式を利用することを宣言し, CafeOBJ コマンド close でその利用を終了することを宣言する. CafeOBJ コマンド red は, 等式を左から右への書き換え規則とみなし, 与えられた項を簡約する.

次にすべての遷移規則に対して帰納段階の証明を行う. 帰納段階では, 各作用演算で表される遷移規則が表明の性質を保持することを示すとき, 状態空間を複数に分割する (場合分けを行う). 分割した各空間 (各場合) ごとに以下のような証明譜を記述する.

```
open ISTEP
(任意の値を表す定数の宣言)
(場合を表す等式の宣言)
(補題等からの事実を表す等式の宣言)
eq s' = a(s,...) .  -- 事後状態
red istep100(cm) .
close
```

まず, 作用演算 a で使われる任意の値を表す定数ならびに議論中の場合を表す等式を宣言する. また, 必要であれば補題等からの事実を等式として宣言する. 続いて宣言する等式

は, 定数 s' が定数 s で表される状態で作用演算 a に対応する遷移規則が適用された事後状態を表すことを意味する. 最後に, 項 istep100(cm) を簡約する. 簡約の結果が期待通り (この場合は true) であれば, この場合の証明が成功したことを意味する. そうでなければ, この場合に対応する空間をさらに分割しなければならないか, さらなる補題を必要とするかを判断する必要がある.

各証明に対応するソースコードについては付録 C を参照して欲しい.

・(Coq)

表明の検証を行うため, 前述の表明 1,2 を表す定理 inv100, inv200, inv300 を宣言する.

```
Theorem inv100 : (s:System)(n:Network;cm:CipherM)
(nw s n) /\ (ccm cm n)
-> (for cm) = intruder.

Theorem inv200 : (s:System)(n:Network;m,l:nat;sk:Skey)
(nw s n) /\ (In (M4 (P m) (ENCM (P l) sk)) n)
-> (P l) = (key_for sk).

Theorem inv300 : (s:System)(n:Network;m,l:nat;sk:Skey)
(nw s n) /\ (In (M5 (P m) (ENCM (P l) sk)) n)
-> (P l) = (key_for sk).
```

CafeOBJ の場合と同様に inv100 で表される述語の証明について考える. Coq では証明の操作を表すタクティクを用いて検証を行い, それらを記録することによって, 帰納法を用いた検証を記述した証明譜を作成する. まず, 証明モードにおいてタクティクを以下のように書き出し, 状態 s に関する帰納法による検証を始める. s に関する場合分けは自動化され, 場合の数と同数のサブゴールが生成される.

```
Intro.
Induction s.
```

初期状態における場合を示すサブゴールの証明から始まる. 初期状態において証明すべきは以下の証明図である.

```
n : Network
cm : CipherM
H0 : (nw init n)
H1 : (ccm cm n)
============================
 (for cm)=intruder
```

証明譜は以下のようになる.

```
Inversion H0.
   (* H0 を nw の定義とマッチング. H2:(nil Msg)=n 生成 *)
Rewrite <- H2 in H1.
   (* H1 を (ccm cm (nil Msg)) に変える *)
Inversion_clear H1
   (* H1 を ccm の定義とマッチング. 仮定が False になり, 証明終了 *)
```

初期状態の証明後, すべての遷移規則に対しての帰納段階を示すサブゴールの証明に移行する. 帰納段階 (の一つ) において証明すべきは以下のような証明図である.

```
s : System
Hrecs : (n:Network; cm:CipherM)(nw s n)/\(ccm cm n) ->(for cm)=intruder
(任意の値を表す定数の宣言)
(場合を表す項の宣言)
n : Network
cm : CipherM
H0 : (nw (a s ...) n)
H1 : (ccm cm n)
============================
  (for cm)=intruder
```

作用演算 a で使われる任意の値を表す定数ならびに議論中の場合を表す項が仮定として現れる. (a s ...) は, 定数 s で表される状態で作用演算 a に対応する遷移規則が適用された事後状態を表す. 証明譜は概ね以下のような形になる.

```
Inverion H0.
   (* H0 を nw の定義とマッチング. 場合分け H2, n に関する仮定 H3 生成 *)
Rewrite <- H3 in H1.
Inversion_clear H1.
     .
(簡約操作)
     .
Apply (Hrecs ...).
    (* 帰納法の仮定を利用してゴールを導く*)
```

簡約中に場合分けが必要なときは、自動的に場合の数と同数のサブゴールが生成され, 場合に対応する空間は分割される. 簡約がうまくいけばこの場合の証明が成功したことを意

味し, 自動的に次の場合を示すサブゴールの証明に移る. そうでなければさらなる補題を見つけ出す必要がある.

各証明に対応するソースコードについては付録 F を参照して欲しい.

第8章　考察

例題として取り上げたSTSプロトコルの形式化と検証を通して, CafeOBJおよびCoqを用いた形式手法の特徴について考察する. 具体的には, システムの形式化および検証の手法についての比較を行い長所・短所を明らかにする.

8.1　システムの形式化と記述

システムの形式的記述の面に関して双方の特徴を考察する.

8.1.1　データ型

CafeOBJ仕様とCoq仕様のそれぞれを作成する際にシステムのモデル化に使用するデータ型の定義を行ったが, 双方の間には「データ型」の表現方式に違いがある.

CafeOBJ仕様では, データ型の定義は一つのモジュールという単位で記述される. そこでは, データ型は数学的な代数として集合とその上で作用するいくつかの演算で定義される. これは, データとそれに関わる操作を一体化しオブジェクトとしてとらえるオブジェクト指向の考えが基になっている. また集合は型分けされ(多ソート), その集合には包含関係が許される(順序ソート). そのため, 定数・関数・述語といった演算の拡張・制限・再利用を容易に行うことができる.

Coq仕様ではデータ型は一つの集合としてとらえられ, 帰納的に定義される(再帰的データ型). データ型に関する各演算の定義は帰納的述語等により別途行う必要がある. Coq仕様において集合は一つの独立したものであり, 部分集合や他集合との共有部分を持たない. これは, すべての対象物(表現・項)はただ一つの型を持ち重複は許さないという型付けられた集合論の考えが基になっている. データ型自体も`Set`と呼ばれる型を持ち, その集合に属している. しかし, そのデータ型の要素が`Set`の集合に含まれているわけではない.

8.1.2　記述法と記述量

一般的にCafeOBJ仕様では自然言語ほどではないものの, システムの記述をオブジェクトの意味を直接捉える方程式の集合によって記述するという直感的な形で行うことができ, 仕様として可読性の高い形式化を行うことができる. しかしそのため, 記述が多少冗

長になる部分もある. 一方 Coq では, システムを集合論と述語論理に基づいた構造化された論理体系として捉えるため記述は型制約などの制限が多く, 直感的な形式化を行うことはできない. また, 構文論的な記号の記述となるため可読性も低くなる. しかしその分, 記述量を抑えることができる.

8.1.3 STS プロトコルの記述

観測遷移機械 OTS で作成した STS プロトコルのモデルの記述を双方で行った際, 以下のような点で違いが見られた.

- 遷移規則の記述
  CafeOBJ 仕様では条件付き等式の条件として効力条件を, 等式として観測値の変化を記述することができる. 一方 Coq 仕様では, スキーマとして記述する. 効力条件は一つの述語として表せるが, 観測値の変化の記述には二つの述語が必要になる.

- 観測演算の記述
  CafeOBJ 仕様では遷移規則を表す条件付き等式の集合で定義する. メッセージ送信を表す作用演算毎に逐次記述を行うことができる. 一方, Coq 仕様では遷移規則を表すスキーマをルールとして持つ帰納的述語として一括に記述し, 定義する.

- 効力条件の記述
  CafeOBJ 仕様では, メッセージを構成する要素 (データ構成子が引数として取るデータ型の要素) についての条件を一つずつ記述し, and でつなぐことによって記述する. そのため冗長となる. 一方 Coq 仕様では, ある形のメッセージがネットワークに存在することを表す存在記号を用いた一つの述語で書ける.

- 侵入者が得ることができる情報の定義の記述
  CafeOBJ 仕様では条件付き等式の集合で記述し, 定義する. 一方, Coq 仕様では帰納的述語として定義する. CafeOBJ 仕様では条件が多くなると冗長になり, Coq 仕様では意味が多少捉えにくい.

仕様書としての可読性を考えた場合, プロトコルの動作の把握がしやすいような記述を行える CafeOBJ 仕様の方がやはり優れている. Coq 仕様では帰納的述語による意味の記述が直感的でなく, その記述と理解に多少の慣れが必要となる. また検証する性質を簡潔に表すため, データ型の定義に含めることができなかった演算をいくつか定義する必要がある. しかし構造的な記述によって記述量を減らすことができ, 効力条件の記述のように存在記号を用いることによってより簡潔に表すことができる部分もあった. そのため, システム全体の構造の把握がしやすい記述を行うことができた. 記述量は本研究での STS プロトコルの記述においては, 性質の記述を含めて CafeOBJ では 21k バイト, Coq では 16k バイトであった.

8.2 システムの検証

形式手法ではシステムがある性質を有していることの検証は, 性質を表す記述が正しいことを処理系で示す (証明する) ことによって行う. そのことに関して双方の特徴を考察する.

8.2.1 処理系で実行する証明譜の特徴

CafeOBJ 処理系では, 事実を表す等式集合と簡約を行う項といったいわば証明そのものを記述した証明譜の実行を行うのに対し, Coq の証明モード[1] では, タクティクと呼ばれるコマンドの系列という一連の証明構築操作を記述したいわばスクリプトのような証明譜の実行を行うといった特徴がある. 前者は既存のテキスト・エディタを用いて証明を閲覧し, 理解したうえで修正を行えることから可読性・編集容易性の面では優れている. その一方, 等式の追加などの証明譜の変更が検証にどの程度影響を与えるのかわかりづらく, 証明譜の一部の書き換えと処理系での実行を何度も繰り返して行う必要がある. 後者は対話的に証明を進められる面や証明譜が一つのゴールの証明に特化されるものではないことから, 証明を含む多くのライブラリを利用することができるなどの証明の再利用の面では優れている. その一方証明譜の主目的は証明の再演であり, 人間が読むためのものではないため可読性が低く, その修正には処理系での実行を伴う必要があるため編集の自由度も低い.

8.2.2 処理系で行われる検証の特徴

CafeOBJ 処理系の検証の基本機構は一方向の等式推論であり, その公理系も等式集合で体系化されたシステムの仕様といった捉えやすいものであるため, 検証系は複雑になり過ぎない. そのため特に定理証明を専門としない者にとっては, 検証を理解が容易な形で行うことができる. その一方, 場合分けはある程度規則的に行えるものの検証を行う者の経験に依存する. そのため, 場合を尽くさないまま検証を終えてしまうという誤りを犯す可能性がある. また, 書き換え規則の適用の手順によって一般に書き換えの経路は複数存在するが, CafeOBJ 処理系では処理系のルールに従って選択された一つの経路の書き換えが行われるだけである. そのため書き換えが意図した経路とは異なる方向に進むことによって, 期待した通りの結果が得られないこともある.

Coq 処理系では場合分けや帰納法の仮定の生成等は, 定義から生成される帰納法の原理および推論規則に従って自動的に行われる. そのため, 場合を尽くさないなどの人間による誤りを含まない検証を行うことができる. その一方, 検証の基本機構は証明図の構築であり, その公理系も推論規則で表される直感主義論理および定義により拡張された論理の形式体系という専門色の強いものであるため, 検証系の理解に定理証明に関する知識を必

[1] 以後 Coq 処理系と呼ぶこととする.

要とする. また検証中に誤ったコマンドを使用してしまうと, 余計な仮定やサブゴールが生成されてしまい検証の方向性を見失う危険性がある. そのため, 適切な仮定やコマンドの選択にある程度実際の経験が必要となる.

8.2.3 STS プロトコルの検証

STS プロトコルの検証を双方の処理系を用いて行った際, 以下のような特徴が見られた.

場合分け・補題の発見

簡約により期待した通りの結果が得られないとき, 新たな場合分けや補題が必要である場合が多い. 場合分けや補題を発見し, 簡約を進めることによって期待した通りの結果が得られる. このことに関して CafeOBJ 処理系と Coq 処理系を比べたとき, Coq 処理系の方が容易であると言うことができる.

CafeOBJ 処理系で補題を発見するには, 簡約後の項の `xor` でつながれた部分項に注目する. この部分項の真偽を考えることによって, 場合分けや補題を発見する. ここで, その項の真偽が場合分けであるのか補題であるのかを注意深く決定する必要がある. なぜなら, 補題という真である項の真偽に関する場合分けを行うことは間違った証明を行うことであるからである. CafeOBJ 処理系での証明では場合分けが多く, 上記の考慮を行う場面がかなりみられる. また, 書き換えの経路がうまく進んでいないために期待した通りの結果が得られない場合のことも考える必要がある. この場合には新たな場合分けや補題を発見するのではなく, トレース結果を遡っていくことによって正しい経路を発見する必要がある.

Coq 処理系では, 場合分けはデータ型に関する場合分けである帰納法に限られるため, 場合分けと補題の区別に迷うことはなく補題の発見を場合分けと切り離して考えることができる. Coq 処理系では, 帰納法の仮定や補題などの仮定とゴールに注目し, 簡約を進めることができる書き換えを一つ発見することで補題を発見できる. その書き換えがその他の仮定から成り立つこと (を示すこと) が補題 (の証明) である. うまく発見できない場合は `Undo` コマンドにより, 簡約を一ステップ戻ることを繰り返すによって発見できる.

帰納段階の証明の省略

表明および補題の検証を行う際に証明譜の記述量を減らすため, 明らかに条件に合わない帰納段階の証明を省略するという方法が考えられる. 例えば補題 3 は正規の Message1 に関する補題なので, 正規の Message2-5 に関する帰納段階や偽造メッセージに関する帰納段階の証明を省略することができる. また補題 1(2) の場合には, 正規および偽造の Message2(Message3) に関わる帰納段階のみを証明すればよいということになる. このことに関して CafeOBJ 処理系と Coq 処理系を比べたとき, CafeOBJ 処理系の方が容易であると言うことができる.

CafeOBJ 処理系では, 表明や補題の検証において関わりがあると考えられる作用演算のみを取り上げて帰納段階の証明を行えばよい. また, そのように検証した補題等を表明や別の補題の検証の際に用いることも問題なく行える.

一方, Coq 処理系ではそのようなことはできない. Coq 処理系において帰納法を用いた検証を行う場合, 帰納段階の網羅的な証明を要求するからである. 補題 2 の場合, 正規および偽造の Message3 に関する帰納段階の証明を行うには, 集合 `System` の定義から結局すべての帰納段階について証明を行う必要があり, 省略することができない. 補題 1 の場合, 偽造の Message2 に関する帰納段階の証明後, `Abort` コマンドで証明を終了して偽造の Message3 に関する帰納段階の証明の部分だけ省略することは可能である. また, 補題 3 では正規の Message1 に関する帰納段階の証明後, `Abort` コマンドにより希望通りの省略を行える. しかし, このように証明を途中で終わらせた補題等をそのまま表明や別の補題の検証に用いることはできない. その際には `Hypothesis` を用いて別の名前の仮定 (未証明の補題) として宣言を新たに行う必要が生じる. そのため帰納段階の証明の省略を行うメリットがほとんどなく, 逆に証明譜を分かりづらくしてしまう.

仕様の変更に伴う証明譜の変更

システムに新たな操作が加わる等の仕様の変更が起こった場合, 変更前に作成した証明譜を修正する必要が生じる. このような場合, 仕様の変更によって検証がどのような影響を受けるのか理解する必要がある.

CafeOBJ 処理系の場合, 証明譜を再度実行することによって変更の影響を受ける部分を知ることが出来る. それは簡約の結果が `true` でなくなった部分である. 証明譜は各場合ごとに分けられているため, 検証者は証明譜のその部分を見ることによって, どのような場合 (空間) が変更の影響を受けたのか理解できる. あとはその場合 (空間) において簡約結果が `true` になるように規則の追加と簡約を繰り返し行い, 証明譜を修正すれば良い. ただし仕様変更によって新たな帰納段階が必要になる場合には, その場合の証明を行うかあるいは省略するかを忘れずに考慮する必要がある.

一方 Coq 処理系の場合, そのようなことはできない. 仕様変更によって定義が変更されると, 論理を拡張する帰納法の原理・推論規則も変わってしまい簡約において生成される仮定の名前などが異なってくるため, 新たな証明構築操作の手順を表す証明譜を再度作る必要が生じるからである. そのためには証明譜に書かれたタクティクを最初から一つ一つ確かめ, 修正を加えていかなくてはならない. 変更の際に誤りを含んでしまうことはないが, かなりの手間が掛かる.

第9章　関連研究

9.1　Bolignanoの方法によるセキュリティプロトコルの検証

本研究ではCoqを用いる方法でセキュリティプロトコルの形式的な検証を行う際, 比較のためCafeOBJを用いる方法の場合と同じく, 観測遷移機械OTSを用いて振る舞いをモデル化したものを記述することによって形式化・検証を行った. Bolignano[6, 7]は本研究とは異なった方法を用いてセキュリティプロトコルのモデルを作成し, Coqによる形式化・検証を行っている. 結果として本研究とは記述および検証に違いが見られる.

9.1.1　セキュリティプロトコルのモデル化・形式化

Bolignanoの方法では, 特定の二人の主体のネットワークを介してのメッセージのやり取りと侵入者が手に入れることのできる情報との関係を考えることにより, セキュリティプロトコルのモデルを作成する.

プロトコル全体の状態は二人の主体およびネットワークの状態で表される. 二人の主体の状態はその通信を行うにあたって保持する必要がある情報の集合によって表される. 一方, ネットワークの状態はメッセージを構成する要素のリストで表現される. ネットワーク上の情報を自由に使用することができる侵入者が手に入れることのできる情報は暗号の偽造・復号を含めた集合で考え, このネットワークの状態を引数に取る帰納的述語として記述し, 定義する. メッセージのやり取りというプロトコルの動作によりそれぞれの状態のうち少なくとも一つは変化し, 結果としてプロトコルの状態は変化する.

メッセージの送信では, 送信者の状態変化とともにメッセージの追加というネットワークの状態変化が起こる. 一方メッセージの受信では, 受信者の状態変化とともに受信者が受け入れるメッセージを侵入者が手に入れることができる情報に追加する. ここで受信者が受け入れるメッセージは, メッセージのフォーマットと受信前の受信者の状態によって決定されるものである. このように捉えたメッセージのやり取りをプロトコルの状態変化を表す演算として記述し, 定義する.

9.1.2　セキュリティプロトコルの検証

状態変化を起こすプロトコルの動作は形式化の節で述べたようにメッセージ数の二倍の演算で表される. そのいずれの演算による状態変化によっても, 性質が失われないことを

証明することによって検証を行う.

9.1.3 STS プロトコルの形式化と検証

Bolignano の方法を用いて STS プロトコルの形式化と検証を行った. 形式化と検証の結果の全体を示すソースコードについては付録 G を参照して欲しい.

形式化

メッセージを表すためのデータ型として, 任意の型 D, 暗号鍵 K, 公開値 E の三つを定義する. 暗号鍵には秘密鍵 (公開鍵) とセッション鍵の二つを考える. メッセージの集合 C はこれらの情報と暗号鍵による暗号文の連なりであると定義する. 以下にそれを示す.

```
Parameter D: Set.
Parameter K: Set.
Parameter E: Set.
Inductive B: Set := K2B: K -> B | D2B: D -> B | E2B: E -> B.
Inductive C : Set :=
Encrypt: C -> K -> C | Pair: C -> C -> C | B2C: B -> C.
Parameter KeyX: D -> K.         (* 秘密鍵 *)
Parameter KeyAB: E -> D -> K.   (* セッション鍵 *)
```

このようなメッセージが流れるプロトコル全体の状態 GlobalState を表すものとして, ネットワークの状態 SS および主体 A(B) の状態 APEXbW(BPEXaW) を表すデータ型を定義する. 主体の状態は, Diffie-Hellman パラメータ, 通信相手の公開値, および実際に鍵交換を行った相手の三つの要素で表される. また, ネットワークから侵入者が手に入れることのできる情報の集合を表す述語 known_in を定義する. known_in 以外を以下に示す.

```
Syntactic Definition SS := (list C).

Inductive AState: Set :=
  APEXbW : D -> E -> D -> AState.
Inductive BState: Set :=
  BPEXaW : D -> E -> D -> BState.
Inductive GlobalState: Set :=
  ABI: AState -> BState -> SS -> GlobalState.
```

各々のメッセージの送受信による状態変化を表す演算 rel1-6 を定義する. ここでは Message2 の送信を表す rel3, および受信を表す rel4 について取り上げる.

```
Parameters Aid, Bid: D. (* 主体A,Bの秘密の値 *)
Definition Triple := [c1, c2, c3: C] (Pair c1 (Pair c2 c3)).

Definition rel3 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases stb1 of (BPEXaW p1 exa1 _) =>
   s2=(cons (Triple (B2C (E2B (Expo p1 Bid))) (B2C (K2B Pb))
                    (Encrypt
                    (Encrypt (Pair (B2C (E2B exa1))
                             (B2C (E2B (Expo p1 Bid)))) Sb)
                    (KeyAB exa1 Bid))) s1) /\
   sta1 = sta2 /\ stb1 = stb2 end end.
```

Message1 より受け取ったパラメータ p1 と主体 A の公開値 exa1 を利用して, ネットワークに情報を追加する. Pb,Sb はそれぞれ主体 B の公開鍵, 秘密鍵 (署名) を表すものとする. 主体 B の状態はここでは変化しない.

```
Definition rel4 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases sta1 sta2 of (APEXbW p1 _  _)
                     (APEXbW p2 exb2 w2) =>
   (known_in (Triple (B2C (E2B exb2)) (B2C (K2B Pb))
                     (Encrypt
                     (Encrypt (Pair (B2C (E2B (Expo p2 Aid)))
                                    (B2C (E2B exb2))) Sb)
                     (KeyAB (Expo p2 Aid) w2))) s1) /\
   stb1 = stb2 /\ s1 = s2 /\ p1 = p2 end end.
```

主体 A が受け入れるメッセージを侵入者が手に入れることのできる情報の集合に加える. 受け入れるメッセージは, 主体 A が保持している Message1 で送信したパラメータ p1 および主体 B の公開鍵と署名によって以上のように表される. また, 主体 A は新たな情報として通信相手の公開値 exb2 を手に入れるため状態変化する.

検証

形式化に続き, 検証を行う. ここでは主体 A が交換した公開値が間違いなく主体 B のものであるという性質について考える. 以下のように記述する.

```
Definition rel := [st1, st2: GlobalState]
  (rel1 st1 st2) \/ (rel2 st1 st2) \/ (rel3 st1 st2) \/ (rel4 st1 st2) \/
  (rel5 st1 st2) \/ (rel6 st1 st2).

Definition CipherM1 := [st: GlobalState]
  Cases st of (ABI (APEXbW p exb _) _ _) => exb = (Expo p Bid) end.
```

その後, いかなるメッセージの送受信による状態変化 (`rel st1 st2`) によっても性質が失われないことを示す定理 `inv100` の証明を行う.

```
Theorem inv100: (st1:GlobalState)(st2:GlobalState)
(CipherM1 st1) -> (rel st1 st2) -> (CipherM1 st2).
```

9.1.4 本研究の方法との比較

形式手法として Coq を用いた検証に関して Bolignano の方法を本研究の方法と比較し, その特徴について述べる.

プロトコルではルールに則ってメッセージのやり取りを行う. Bolignano の方法ではこのやり取りをメッセージごとに送信と受信とに分け, プロトコル全体の状態変化という形で記述する. ルールは主に受信者の状態と受信するメッセージの関係で記述する. ルールとして効力条件を別に記述しメッセージの送信だけを考える本研究に比べ, 記述からプロトコルの仕様を読み取ることは難しいと言える.

メッセージ送信は双方とも, メッセージの集合であるネットワークにメッセージを追加することによって表す. 双方の間にはメッセージを表すデータ型に関する違いがある. Bolignano の方法では, メッセージはプロトコルで使用するデータ型の要素の任意の連なりで表される. 各々のデータ型には型を一致させるための型変換が必要となる. 各メッセージごとに特定のフォーマット (コンストラクタ) を持ったものの集合によって定義する本研究に比べ, このことからも記述からプロトコルの仕様を読み取ることは難しいと言える.

侵入者の振る舞いの記述は双方とも, 侵入者が手に入れることができる情報を表す帰納的述語として記述する. Bolignano の方法では暗号文の復号や作成などの侵入者の行動を, 行動によって得ることができる情報としてこの帰納的述語に含めて定義する. 侵入者の行動を偽造メッセージの送信という形で多くの場合を考えて記述する必要がある本研究に比べ, 簡潔に書くことができる. しかしその分, 検証において証明の構造はさらに捉えにくくなる.

Bolignano の方法で作成したプロトコルのモデルは Coq で直接記述することができる. そのため, モデル化と形式化をほぼ同時に行うことができる. また, 異なるプロトコルの性質の検証においても似通った証明を行える. 先ほどの定理 `inv100` の証明では, [7] 内の Otway-Rees プロトコルの検証を行った証明譜を参考にすることができた. このように Coq

の性質を生かし, Coq に適した形式化および検証を行えることから手法としては有用である. しかし上述のように形式化と検証がわかりづらく可読性も低いため, その意味においては本研究の方法の方が優れていると言える.

9.2 Coq によるプログラムコード抽出

証明支援系 Coq では定理の証明を半自動的に行えるだけでなく, 証明からプログラムコードを抽出する[1] こともできる. システムの形式化から, 検証およびコードの抽出を行う形式手法によるプログラム開発は誤りを含まないことに対して高い保証を与えることができると期待されており, 正しさが求められるプログラム (型推論器など) のコード生成に利用, 研究されている.

本研究の Coq を用いた STS プロトコルの検証においていくつかの修正を加えることにより, この仕様段階におけるプログラムコード抽出を行うことができる. 具体的には, 記述の中から侵入者の振る舞いの関する部分を削除しいくつかの定理の証明を Coq 上で行う必要がある. 例として, 任意の状態において観測値である送信済みメッセージの集合が存在することを表す定理を証明することにより, (初期状態への任意回の推論規則の適用で表される) ある状態における (状態空間に関連する値である) 通信網を介して送信済みであるメッセージの集合を求めるプログラム `fun_nw` を抽出した.

例えば, 状態 (`mes2 (mes1 init p1 p2 pa) p1 p2 pa co`) を引数に取れば

```
(cons (M2 p2 p1 (COMP pa (RAND p2)) (CERT p2 pa)
                (ENC (SKEY co (RAND p2))
                     (SIGN p2 (COMP pa (RAND p2)))))
      (M1 p1 p1 p2 pa (COMP pa (RAND p1))) nil)
```

というリスト (ネットワーク) を返す. 抽出した Objective Caml のソースコードについては付録 H を参照して欲しい. このプログラムは, STS プロトコルの解析やシミュレーションを行う際に利用することができる.

また, プロトコルのリアルタイム性に関わる部分等のプログラムコードを抽出したコードに拡張という形で付け足せすことにより, プロトコルに関する実際のソフトウェア開発の中でも利用できる可能性がある. `fun_nw` の抽出を行った際, 他に以下のような関数も抽出された.

[1] Haskell[21], Objective Caml[22] などの関数型プログラミング言語のソースコードを抽出する.

- `fun_cmes2`
  - 引数　　ネットワーク `n`, 主体 `p`,p21, D.H. パラメータ `pa`, 公開値 `co`
  - 機能　　Message2 発信の条件判定
  - 返り値
    n に p1 から p2 向けの pa および co を用いた Message1 があれば Left.
    なければ Right.

- `fun_cmes3`
  - 引数　　ネットワーク `n`, 主体 `p1,p2`, D.H. パラメータ `pa`, 公開値 `co1,co2`
  - 機能　　Message3 発信の条件判定
  - 返り値
    n に p1 から p2 向けの pa および co1 を用いた Message1, およびそれに対応する p2 から p1 向けの co2 を用いた Message2 があれば Left. なければ Right.

- `fun_cmes4`
  - 引数　　`fun_cmes3` と同じ
  - 機能　　Message4 発信の条件判定
  - 返り値　`fun_cmes3` と同じ

- `fun_cmes5`
  - 引数　　ネットワーク `n`, 主体 `p1,p2`, D.H. パラメータ `pa`, 公開値 `co1,co2`
  - 機能　　Message5 発信の条件判定
  - 返り値
    n に p1 から p2 向けの pa および co1 を用いた Message1, それに対応する p2 から p1 向けの co2 を用いた Message2, およびさらにそれに対応する p1 から p2 向けの Message3 があれば Left. なければ Right.

これらの関数を STS プロトコルの鍵交換を実現するクライアント上のソフトウェア開発に用いることが出来る. 具体的には, メッセージの作成や暗号の復号および通信網を介した送受信などの実際の操作・通信を行うプログラムと, 送受信されるメッセージを上記の関数に用いることの出来る形[2]に変換するプログラム `trans` を作成 (あるいは抽出) し, ネットワーク `n` および鍵交換に使用する D.H. パラメータや公開値などの情報を格納するメモリ空間を用意すれば良い.

[2]例えば `(M1 p1 p1 p2 pa (COMP pa (RAND p1)))` など

その上で, クライアント上のソフトウェアにおいて以下のように使用する.

- メッセージを受信するとき
  1. trans を用いて, 受信したメッセージを変換する.
  2. ネットワーク n にそのメッセージを追加する.
  3. 次に行うべきメッセージ送信を行えるかどうかを fun_cmesx を用いて判定する.
- メッセージを送信するとき
  1. trans を用いて, 送信するメッセージを変換する.
  2. ネットワーク n にそのメッセージを追加する.
  3. 通信網を介してメッセージを送信する.

このように型推論器のような静的なシステムだけでなく, プロトコルのような動的なシステムの開発にも有用な手法として, 研究を行う価値があると思われる.

第10章　まとめと今後の課題

10.1　まとめ

セキュリティプロトコルの一つであるSTSプロトコルの振る舞いを観測遷移機械OTSを用いてモデル化し, CafeOBJ仕様およびCoq仕様で記述, 形式化した. またCafeOBJ処理系およびCoq処理系において, 安全性と信頼性を示す性質を表す記述を証明することにより, それぞれの仕様の検証を行った. さらに双方を比較することによって, それぞれを用いた形式手法の特徴を捉えることができた.

一般にソフトウェアの開発は上流工程である仕様記述から外部設計, 内部設計, プログラム設計, 下流工程である最終的なプログラムコードの作成, テストまでの段階を経て行われる. 上流工程である仕様記述の段階で欠陥のあるシステムを開発してしまった場合, システム運用上での社会的・経済的損失は免れない. システムが大規模になればその損失は大きなものになる. そのため, 仕様記述の段階で誤りのないことを保証するソフトウェア検証は非常に重要であり, 形式手法によるもの[1]もその一つである.

システムの基本計画を行う仕様記述の段階では, 検証の知識を持たない人が多く関わることや, システムの機能の追加などの仕様変更およびそれに伴う証明譜の変更がたびたび起こる可能性があるため, 仕様段階における形式手法としては可読性が高くその構造が分かりやすい仕様および証明譜の作成が可能である手法ほど適していると言える.

また, 形式手法によるソフトウェア検証には一般的に非常に大きいコストが必要になるとされている. ここでコストとは, 仕様の記述および検証に時間がかかるという時間的要因と, 仕様の記述者および検証者にその方法論の理解とある程度の経験を求めるという人材の育成 (熟練度) に関わる要因によって決まるものと考える. 形式手法としては当然, 出来る限りコストの掛からない手法が良いと言える.

OTSにより作成されたシステムのモデルに対するCafeOBJを用いた形式手法 (OTS/CafeOBJ) は仕様記述が可読性に優れ, 現在ソフトウェア開発・設計において主流となりつつあるオブジェクト指向設計に通じるものがあり仕様書としても有用であることや, 検証系の理解が比較的容易で証明譜の可読性も高いことがその特徴である. これらの特徴により, 仕様段階における検証手法として適していると言うことができる. またOTSで作成されたモデルを直接条件付き等式の形で構造的に記述できるため, 小さいコストで仕様の記述を行うことができる.

[1]以後, 仕様段階における形式手法と呼ぶ

CafeOBJ 処理系における検証は, 項の書き換えにより記述が正しいことを証明することによって行われる. 検証者は場合分けという形で書き換えのルールをいくつか追加し処理系で項を簡約する命令を与えれば, 書き換えが進み (証明が成功したか否かの) 結果が得られる. このとき, 検証者は書き換えの詳細を知る必要はない. このことにより, 結果が明らかである証明を一コマンドで終えることができる可能性がある. また, 逆にどうなるのか見当が付かない証明や仕様変更に伴って再度行う必要の生じた証明を進めることもしくは終えることができる可能性がある. つまり, 検証者は実際の簡約操作を行うことなしに証明を行うことができ, コストを抑えることができる. さらに検証を必要な部分だけ行うことも可能であり, 結果が明らかなものをその対象から外すことによってもコストを抑えることができる.

一方で, 逆に大きいコストが掛かってしまう可能性もある. システムが複雑になると, 検証者が行う必要がある帰納法に関するものも含めた場合分けが多くなる. さらに, 適切な場合分けや補題の発見が検証者に依る部分も大きくなる. その結果, コストは増大する. また処理系によって選択された書き換え規則の適用の手順によっては, 期待した結果が得られない可能性がある. その場合には, 原因の発見にはどの程度進んでしまっているかわからない書き換えの経路を逆にたどっていく必要があり, 非常に大きいコストが掛かってしまう.

Coq を用いた形式手法では, 人間による誤りを含まない厳密的な検証を行うことができる. その反面, 一般的に形式化および検証の結果 (Coq 仕様・証明譜) は可読性に乏しいものが多く, 仕様段階における形式手法としてはあまり適していない技法と言える. しかし, 本研究で示した OTS により作成されたシステムのモデルに対する Coq を用いた形式手法 (OTS/Coq) では, 可読性を大きく失うことなくシステムを形式化することができる. そのため, 仕様段階における形式手法としても十分使用することができる手法であると考えるが, 証明譜の可読性はやはり低いため専門家でない第三者が検証の過程を理解することは難しい. 仕様記述の作成には, 集合論・述語論理に関する知識が必要なため OTS/CafeOBJ に比べコストは大きくなるが, 記述量を抑えることができるという長所もある.

Coq 処理系における検証は定理を証明することによって行うが, 以下のような理由からそのコストは大きい. 検証者には, その定理を手で証明図を書くことによって証明することができる, もしくは少なくともその証明図を理解することができるくらいの定理証明に関する知識と問題に対する理解が必要となる. そのため, 定理証明を専門としない者にとっては検証を行うことは難しい. また, Coq はあくまで人間が手で証明図を書くことを支援し, 仮定やゴールなどの書き換えといったルーチンワークや場合分けなどの厳密性が必要な部分を人間に代わって機械的に行うだけである. そのため, 検証者は一つ一つコマンドを指定しながら簡約を進めて行く必要がある.

一方で, 高階論理を含めた論理学を学び, 推論規則の適用により証明図を描くことによって定理の証明を行う技法に慣れた者であれば, 場合分けに頭を使う必要がないことや対話的に自分の意図通りの検証が行えることから逆に取り組みやすく, 検証のコストを最小限に抑えることができる. また関連研究で述べたプログラムコードの抽出を行うことで, 仕

様段階においてシステムの核となる部分のコーディングを含めた形式手法として Coq を取り入れることによって, コストに見合うものなる可能性がある.

以上のことから次のような結論を出す. CafeOBJ を用いる場面としては, 検証者以外が形式化や検証の過程および結果を見たり理解したりする必要があるときや、システムの仕様が流動的である場合が良いと考える. 検証や検証者の育成に多くの時間を費やせないときにも利用してみる価値はある. 一方 Coq を用いる場面としては, 場合の数が多くなるシステムに対し誤りなく検証を行いたいときや, 完成したシステムが求められる性質を満たしているかどうか知りたいようなときに専門家による検証を行う場合が良いと考える. 検証からコード抽出というプラスアルファを得たい場合にも利用してみる価値はある.

10.2 今後の課題

本研究の今後の課題としては次のようなことが考えられる.

- システムの検証に関する手法比較
  本研究では代数仕様言語および証明支援系を用いた形式手法による検証の比較を行うため, それぞれ CafeOBJ を用いる方法と Coq を用いる方法で形式化と検証を行い, 双方の比較を行った. 今後の課題の一つとして, Maude[19] や Isabelle/HOL[16, 20] などといった CafeOBJ や Coq とは別のツールによる代数仕様言語および証明支援系を用いた形式手法や, 本論文では取り上げなかったモデル検査や様相論理などを用いた他の形式手法も比較の対象として取り上げ, 形式手法に関するさらなる考察を行うことが挙げられる.

- 現実的なプロトコルへの適用
  本研究において行った STS プロトコルの形式化と検証という例題を通して得られた知見を基にして, 認証プロトコル一般に対する形式化と検証の方法論を確立させ, SSL, SSH などの現実的に利用されているようなプロトコルに適用させることも今後の課題の一つである.

謝辞

本研究を進めるにあたりご指導いただいた二木厚吉教授に深く感謝いたします. 緒方和博客員研究員, 中村正樹助手には本研究についての有益な御助言をしていただきました. 心より感謝いたします. また, ゼミを通して研究に関する議論につきあっていただいた言語設計学講座の皆様に感謝いたします.

参考文献

[1] W. Diffie, P. van Oorschot and M. Wiener, *Authentication and authenticated key exchanges*, Designs, Codes and Cryptography, 2, 107-125, 1992.

[2] W. Diffie, and M.E. Hellman, *New directions in cryptography*, IEEE Trans. On Information Theory, Vol.IT-22, No.6, 644-654, 1976.

[3] Kazuhiro Ogata, Kokichi Futatsugi, Rewriting-Based verification of authentication protocols. *the 4th International Workshop in Rewriting Logic and it's Applications(WRLA 2002)*, Electoronic Notes in theoretical Computer Science 71, 15pages. Elserver Science Publishers, 2002.

[4] 緒方和博, 二木厚吉, 書き換えによるセキュリティプロトコルの帰納的検証, コンピュータソフトウェア, vol.20, No3, pp.54-72, 2003.

[5] 加藤淳, 代数仕様言語 CafeOBJ によるセキュリティプロトコルの形式化, 北陸先端科学技術大学院大学, 修士論文, 2004.

[6] D. Bolignano, *Formal verification of cryptographic protocols*, Proceedings of the third ACM Conference on Computer and Communication Security, 1996.

[7] D. Bolignano, *Formal verification of cryptographic protocols using Coq*, Technical report, INRIA-Rocquencourt, 1996.

[8] G. Lowe, *An Attack on the Needham-Schroeder Public-key Authentication Protocol*, Information Processing Letters, Vol. 56, 131-133, 1995.

[9] L. C. Paulson, *The Inductive Approach to Verifying Cryptographic Protocols*, Journal of Computer Security, Vol. 6, 85-128, 1998.

[10] S. Schneiser, *Verifying Authentication Protocols in CSP*, IEEE Transactions on Software Engineering, Vol. 24, No. 9, 741-758, 1998.

[11] M. Abadi, M. Burrows, and R. Needham, *A Logic of Authentication*, ACM Transactions on Computer Systems, Vol. 8, No. 1, 18-36, 1990.

[12] G. Denker, J. Meseguer, J. and C. Talcott, *Protocol Specification and Analysis in Maude*, Workshop on Formal Methods and Security Protocols (http://www.cs.bell-labs.com/who/nch/fmsp/), 1998.

[13] G. Lowe, *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*, 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS, Vol. 1055, Springer, 147-166, 1996.

[14] R. Diaconescu, and K. Futatsugi, *CafeOBJ Report*, AMAST Series in Computing, Vol. 6, World Scientific, Singapore, 1998.

[15] A. Nakagawa, T. Sawada, K. Futatsugi, *CafeOBJ User'sManual*, *http://www.ldl.jaist.ac.jp/cafeobj/doc/*, 1999.

[16] T. Nipkow, L C. Paulson, M. Wenzel, *A Proof Assistant for Higher-Order Logic*, *http://www4.in.tum.de/ nipkow/LNCS2283/tutorial.pdf*

[17] CafeOBJ, *http://www.ldl.jaist.ac.jp/cafeobj/*

[18] Coq, *http://pauillac.inria.fr/coq/*

[19] Maude, *http://maude.cs.uiuc.edu/*

[20] Isabelle, *http://www.cl.cam.ac.uk/Research/HVG/Isabelle/*

[21] Haskell. *http://www.haskell.org/*

[22] Objective Caml, *http://caml.inria.fr/ocaml/*

付 録A　STSプロトコルの仕様 (CafeOBJ)

```
-- ----------------------------------------------------------------------
--
-- STS protocol
--
-- Msg1 A --> B : PA, CO_A
-- Msg2 B --> A : CO_B, CE_B, {{CO_B}IK_B}SK_AB
-- Msg3 A --> B : CE_A, {{CO_A}IK_A}SK_AB
--
-- Msg4 A --> B : {private data}SK_AB
-- Msg5 B --> A : {private data}SK_AB
--
-- CE_A = {(A,PK_A),PA}IK_S
-- CE_B = {(B,PK_B),PA}IK_S
--
-- ----------------------------------------------------------------------

--
-- 主体の定義
-- 侵入者は汎用の侵入者としてモデル化する
--
mod* PRINCIPAL principal-sort Principal {
[Principal]
op intruder : -> Principal
op _=_ : Principal Principal -> Bool {comm}
var P : Principal
eq (P = P) = true .
}


--
-- パラメータの定義
--
mod* PARAMETER principal-sort Parameter {
[Parameter]
op _=_ : Parameter Parameter -> Bool {comm}
var PA : Parameter
eq (PA = PA) = true .
}


--
-- 乱数（類推不可能）の定義
-- rand(p) は主体 p が作成した乱数で p のみ使用可
--
mod! RANDOM principal-sort Random {
pr(PRINCIPAL)
[Random]
op rand : Principal -> Random
op pr : Random -> Principal
op _=_ : Random Random -> Bool {comm}
vars RA1 RA2 : Random
```

```
var P : Principal
eq pr(rand(P)) = P .
eq (RA1 = RA2) =  (pr(RA1) = pr(RA2)) .
}


--
-- セッション鍵を作るための構成要素の定義
-- comp(pa,ra) はパラメータ pa と乱数 ra から
-- 生成した構成要素 (所有者は ra の作成者) を表す
--
mod! COMPONENT principal-sort Component {
pr(PARAMETER + RANDOM)
[Component]
op comp : Parameter Random -> Component
op para : Component -> Parameter
op rand : Component -> Random
op _=_ : Component Component -> Bool {comm}
vars CO1 CO2 : Component
var PA : Parameter
var RA : Random
eq para(comp(PA,RA)) = PA .
eq rand(comp(PA,RA)) = RA .
eq (CO1 = CO2) = (para(CO1) = para(CO2) and
                  rand(CO1) = rand(CO2)) .
}


--
-- セッション鍵の定義
-- skey(com,ra) は構成要素 com と乱数 ra より得られるセッション鍵を表す
-- 構成要素 com の所有者および乱数 ra の作成者 (鍵作成者) 間の通信に利用
--
mod! SKEY principal-sort Skey {
pr(COMPONENT)
[Skey]
op skey : Component Random -> Skey
op com : Skey -> Component
op rdm : Skey -> Random
op _=_ : Skey Skey -> Bool {comm}
vars SK SK1 SK2 : Skey
var CO : Component
var RA : Random
eq com(skey(CO,RA)) = CO .
eq rdm(skey(CO,RA)) = RA .
eq (SK1 = SK2) = (com(SK1) = com(SK2) and
                  rdm(SK1) = rdm(SK2)) .

-- rsk(sk) : 侵入者は鍵 sk(侵入者以外が作成) で暗号化された文を複合可能
op rsk : Skey -> Bool .
eq rsk(SK) = (pr(rand(com(SK))) = intruder) .
}


--
-- Message2,3 に現れる証明書
-- cert(p,pa) は主体 p とその公開鍵の組 (認証済み) とパラメータ pa
-- に認証局 S の署名を施した証明書を表す
--
mod! CERTIFICATE principal-sort Certificate {
pr(PRINCIPAL + PARAMETER)
[Certificate]
op cert : Principal Parameter -> Certificate
op pr : Certificate -> Principal
op pa : Certificate -> Parameter
op _=_ : Certificate Certificate -> Bool {comm}
```

```
vars CE1 CE2 : Certificate
var P : Principal
var PA : Parameter
eq pr(cert(P,PA)) = P .
eq pa(cert(P,PA)) = PA .
eq (CE1 = CE2) = (pr(CE1) = pr(CE1) and
                  pa(CE1) = pa(CE2)) .
}


--
-- Message2,3 に現れる署名
-- sign(p,co) は構成要素 co に主体 p の署名を施した署名を表す
--
mod! SIGN principal-sort Sign {
pr(PRINCIPAL + COMPONENT)
[Sign]
op sign : Principal Component -> Sign
op pr : Sign -> Principal
op co : Sign -> Component
op _=_ : Sign Sign -> Bool {comm}
vars SI1 SI2 : Sign
var P : Principal
var CO : Component
eq pr(sign(P,CO)) = P .
eq co(sign(P,CO)) = CO .
eq (SI1 = SI2) = (pr(SI1) = pr(SI2) and
                  co(SI1) = co(SI2)) .
}


--
-- Message2,3 に現れる暗号文
-- enc(sk,si) は署名 si をセッション鍵 sk で暗号化した暗号文を表す
--
mod! CIPHER principal-sort Cipher {
pr(SIGN + SKEY)
[Cipher]
op enc : Skey Sign -> Cipher
op skey : Cipher -> Skey
op sign : Cipher -> Sign
op _=_ : Cipher Cipher -> Bool {comm}
vars CI1 CI2 : Cipher
var SK : Skey
var SI : Sign
eq skey(enc(SK,SI)) = SK .
eq sign(enc(SK,SI)) = SI .
eq (CI1 = CI2) = (skey(CI1) = skey(CI2) and
                  sign(CI1) = sign(CI2)) .
}


--
-- Message4,5 に現れる暗号文
-- enc(p,sk) は主体 p に向けた、セッション鍵 sk で暗号化した暗号文を表す
--
mod! CIPHERM principal-sort CipherM {
pr(SKEY)
[CipherM]
op enc : Principal Skey -> CipherM
op for : CipherM -> Principal
op key : CipherM -> Skey
op _=_ : CipherM CipherM -> Bool {comm}
vars CI1 CI2 : CipherM
var P : Principal
var SK : Skey
```

```
eq for(enc(P,SK)) = P .
eq key(enc(P,SK)) = SK .
eq (CI1 = CI2) = (for(CI1) = for(CI2) and
                  key(CI1) = key(CI2)) .
}


--
-- メッセージのための可視ソート
--
mod! MSG principal-sort Msg {
pr(CERTIFICATE + CIPHER + CIPHERM)
[Msg]
--      creator  sender     receiver  send-message
-- ------------------------------------------------------------
op m1 : Principal Principal Principal Parameter Component -> Msg
--
--      creator   receiver  send-message
-- ------------------------------------------------------------
op m2 : Principal Principal Component Certificate Cipher -> Msg
op m3 : Principal Principal Certificate Cipher -> Msg
--
--      creator   cryptMsg
-- -------------------------------------
op m4 : Principal CipherM -> Msg
op m5 : Principal CipherM -> Msg
--
ops m1? m2? m3? m4? m5? : Msg -> Bool
ops creator sender receiver : Msg -> Principal
op pa : Msg -> Parameter
op co : Msg -> Component
op ce : Msg -> Certificate
op ci : Msg -> Cipher
op cm : Msg -> CipherM
op _=_ : Msg Msg -> Bool {comm}
--
vars M M1 M2 : Msg
vars CP SP RP : Principal
var PA : Parameter
var CO : Component
var CE : Certificate
var CI : Cipher
var CM : CipherM
--
-- 与えられたメッセージがどの種類のメッセージか判定する
eq m1?(m1(CP,SP,RP,PA,CO)) = true .
eq m1?(m2(CP,RP,CO,CE,CI)) = false .
eq m1?(m3(CP,RP,CE,CI)) = false .
eq m1?(m4(CP,CM)) = false .
eq m1?(m5(CP,CM)) = false .
--
eq m2?(m1(CP,SP,RP,PA,CO)) = false .
eq m2?(m2(CP,RP,CO,CE,CI)) = true .
eq m2?(m3(CP,RP,CE,CI)) = false .
eq m2?(m4(CP,CM)) = false .
eq m2?(m5(CP,CM)) = false .
--
eq m3?(m1(CP,SP,RP,PA,CO)) = false .
eq m3?(m2(CP,RP,CO,CE,CI)) = false .
eq m3?(m3(CP,RP,CE,CI)) = true .
eq m3?(m4(CP,CM)) = false .
eq m3?(m5(CP,CM)) = false .
--
eq m4?(m1(CP,SP,RP,PA,CO)) = false .
eq m4?(m2(CP,RP,CO,CE,CI)) = false .
eq m4?(m3(CP,RP,CE,CI)) = false .
```

```
eq m4?(m4(CP,CM)) = true .
eq m4?(m5(CP,CM)) = false .
--
eq m5?(m1(CP,SP,RP,PA,CO)) = false .
eq m5?(m2(CP,RP,CO,CE,CI)) = false .
eq m5?(m3(CP,RP,CE,CI)) = false .
eq m5?(m4(CP,CM)) = false .
eq m5?(m5(CP,CM)) = true .
--
-- メッセージを送信（作成）した主体
eq creator(m1(CP,SP,RP,PA,CO)) = CP .
eq creator(m2(CP,RP,CO,CE,CI)) = CP .
eq creator(m3(CP,RP,CE,CI)) = CP .
eq creator(m4(CP,CM)) = CP .
eq creator(m5(CP,CM)) = CP .
-- メッセージの見かけの送信者
eq sender(m1(CP,SP,RP,PA,CO)) = SP .
-- メッセージの受信者
eq receiver(m1(CP,SP,RP,PA,CO)) = RP .
eq receiver(m2(CP,RP,CO,CE,CI)) = RP .
eq receiver(m3(CP,RP,CE,CI)) = RP .
-- message 中のパラメータ
eq pa(m1(CP,SP,RP,PA,CO)) = PA .
-- message 中の構成要素
eq co(m1(CP,SP,RP,PA,CO)) = CO .
eq co(m2(CP,RP,CO,CE,CI)) = CO .
-- message 中の証明書
eq ce(m2(CP,RP,CO,CE,CI)) = CE .
eq ce(m3(CP,RP,CE,CI)) = CE .
-- message 中の暗号文
eq ci(m2(CP,RP,CO,CE,CI)) = CI .
eq ci(m3(CP,RP,CE,CI)) = CI .
-- 秘密情報を含む暗号文
eq cm(m4(CP,CM)) = CM .
eq cm(m5(CP,CM)) = CM .
--
eq (M = M) = true .
ceq (M1 = M2) = (m1?(M2) and creator(M1) = creator(M2) and
                 sender(M1) = sender(M2) and
                 receiver(M1) = receiver(M2) and
                 pa(M1) = pa(M2) and co(M1) = co(M2)) if m1?(M1) .
ceq (M1 = M2) = (m2?(M2) and creator(M1) = creator(M2) and
                 receiver(M1) = receiver(M2) and
                 co(M1) = co(M2) and ce(M1) = ce(M2) and
                 ci(M1) = ci(M2)) if m2?(M1) .
ceq (M1 = M2) = (m3?(M2) and creator(M1) = creator(M2) and
                 receiver(M1) = receiver(M2) and
                 ce(M1) = ce(M2) and ci(M1) = ci(M2)) if m3?(M1) .
ceq (M1 = M2) = (m4?(M2) and creator(M1) = creator(M2) and
                 cm(M1) = cm(M2)) if m4?(M1) .
ceq (M1 = M2) = (m5?(M2) and creator(M1) = creator(M2) and
                 cm(M1) = cm(M2)) if m5?(M1) .
}


--
-- パラメタつきモジュール宣言に必要なモジュール
--
mod* EQTRIV principal-sort Elt {
pr(TRIV)
op _=_ : Elt Elt -> Bool {comm}
}


--
-- 汎用の多重集合の定義
```

```
--
mod! BAG (D :: EQTRIV) principal-sort Bag {
[Elt.D < Bag]
op void : -> Bag
op _,_ : Bag Bag -> Bag {assoc comm id: void}
op _\in_ : Elt.D Bag -> Bool
var B : Bag
vars E1 E2 : Elt.D
eq E1 \in void = false .
ceq E1 \in (E2,B) = true if E1 = E2 .
ceq E1 \in (E2,B) = E1 \in B if not(E1 = E2) .
}


--
-- 汎用の集合の定義
--
mod! SET (D :: EQTRIV) principal-sort Set {
[Elt.D < Set]
op empty : -> Set
op __ : Set Set -> Set {assoc comm idem id: empty}
op _\in_ : Elt.D Set -> Bool
var S : Set
vars E1 E2 : Elt.D
eq E1 \in empty = false .
ceq E1 \in (E2 S) = true if E1 = E2 .
ceq E1 \in (E2 S) = E1 \in S if not(E1 = E2) .
}


--
-- 汎用のものの集まりを定義
--
mod* COLLECTION (D :: TRIV) principal-sort Col {
[Elt.D < Col]
op _\in_ : Elt.D Col -> Bool
}


--
-- ネットワークの定義
--
mod! NETWORK {
pr(BAG(MSG)*{sort Bag -> Network})
pr(COLLECTION(COMPONENT)*{sort Col -> ColComp})
pr(COLLECTION(SIGN)*{sort Col -> ColSign})
pr(COLLECTION(CIPHER)*{sort Col -> ColCipher})
pr(COLLECTION(CIPHERM)*{sort Col -> ColCipherM})
--
op cco : Network -> ColComp
op csi : Network -> ColSign
op cci : Network -> ColCipher
op ccm : Network -> ColCipherM
--
var CO : Component
var CE : Certificate
var SI : Sign
var CI : Cipher
var CM : CipherM
var M : Msg
var NW : Network

--
-- cco : 侵入者がネットワークから収集できる構成要素
-- 1 番目 : ネットワークが空であれば、侵入者が利用できる構成要素はない
-- 2 番目 : ネットワーク中に Message1 が存在すれば、その中に現れる構成要素
```

```
--           を利用できる
-- 3 番目 : ネットワーク中に Message2 が存在すれば、その中に現れる構成要素
--           を利用できる
-- 4 番目 : Message1,2 以外のメッセージからは該当する (新たな) 構成要素は
--           得られない
--
eq CO \in cco(void) = false .
ceq CO \in cco(M,NW) = true
if m1?(M) and not(creator(M) = intruder) and co(M) = CO .
ceq CO \in cco(M,NW) = true
if m2?(M) and not(creator(M) = intruder) and co(M) = CO .
ceq CO \in cco(M,NW) = CO \in cco(NW)
if not(m1?(M) and not(creator(M) = intruder) and co(M) = CO) and
   not(m2?(M) and not(creator(M) = intruder) and co(M) = CO) .
--
-- csi : 侵入者がネットワークから収集できる署名
-- 1 番目 : ネットワークが空であれば、侵入者が利用できる署名はない
-- 2 番目 : ネットワーク中に Message2 が存在し、その中の暗号文が、
--           侵入者が復号可能なセッション鍵により暗号化されているなら、
--           その中に現れる署名を利用できる
-- 3 番目 : ネットワーク中に Message3 が存在し、その中の暗号文が、
--           侵入者が復号可能なセッション鍵により暗号化されているなら、
--           その中に現れる署名を利用できる
-- 4 番目 : Message2,3 以外のメッセージからは該当する署名は得られない
--
eq SI \in csi(void) = false .
ceq SI \in csi(M,NW) = true
if m2?(M) and rsk(skey(ci(M))) and not(creator(M) = intruder) and
   sign(ci(M)) = SI .
ceq SI \in csi(M,NW) = true
if m3?(M) and rsk(skey(ci(M))) and not(creator(M) = intruder) and
   sign(ci(M)) = SI .
ceq SI \in csi(M,NW) = SI \in csi(NW)
if not(m2?(M) and rsk(skey(ci(M))) and  not(creator(M) = intruder) and
   sign(ci(M)) = SI) and
   not(m3?(M) and rsk(skey(ci(M))) and  not(creator(M) = intruder) and
   sign(ci(M)) = SI) .
--
-- cci : 侵入者がネットワークから収集できる暗号文
-- 1 番目 : ネットワークが空であれば、侵入者が利用できる暗号文はない
-- 2 番目 : ネットワーク中に Message2 が存在すれば、その中に現れる暗号文を
--           利用できる
-- 3 番目 : ネットワーク中に Message3 が存在すれば、その中に現れる暗号文を
--           利用できる
-- 4 番目 : Message2,3 以外のメッセージからは該当する暗号文は得られない
--
eq CI \in cci(void) = false .
ceq CI \in cci(M,NW) = true
if m2?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI .
ceq CI \in cci(M,NW) = true
if m3?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI .
ceq CI \in cci(M,NW) = CI \in cci(NW)
if not(m2?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI) and
   not(m3?(M) and not(rsk(skey(ci(M)))) and
   not(creator(M) = intruder) and ci(M) = CI) .
--
-- ccm : 侵入者が解読できるネットワーク上の暗号通信文の集合
--
eq CM \in ccm(void) = false .
ceq CM \in ccm(M,NW) = true
if m4?(M) and rsk(key(cm(M))) and cm(M) = CM .
ceq CM \in ccm(M,NW) = true
if m5?(M) and rsk(key(cm(M))) and cm(M) = CM .
```

```
ceq CM \in ccm(M,NW) = CM \in ccm(NW)
if not(m4?(M) and rsk(key(cm(M))) and cm(M) = CM) and
   not(m5?(M) and rsk(key(cm(M))) and cm(M) = CM) .
}


-- ---------------------------------------------------------------------
--
-- STS Protocol のモデル
--
mod* STS {
pr(NETWORK)
*[System]*
-- any initial state
op init : -> System

-- observation operations
bop nw : System -> Network

-- for any initial state
eq nw(init) = void .

-- CafeOBJ variables
vars P P1 P2 : Principal
vars M M1 M2 M3 : Msg
vars PA PA1 PA2 : Parameter
vars CO CO1 CO2 : Component
var CE : Certificate
var SI : Sign
var CI : Cipher
var S : System

-- sending message
--  M1 送信者  M1 受信者  パラメータ
bop mes1 : System Principal Principal Parameter -> System
--  M2 送信者  M2 受信者  message1
bop mes2 : System Principal Principal Msg -> System
--  M3 送信者  M3 受信者  message1   message2
bop mes3 : System Principal Principal Msg Msg -> System
--  M4 送信者  M5 受信者  message1  messafe2
bop mes4 : System Principal Principal Msg Msg -> System
--  M5 送信者  M5 受信者  message2  message3
bop mes5 : System Principal Principal Msg Msg -> System


-- for mes1
--
-- mes1 : P1 は P2 と安全な通信を行うべく、message1 を送信
--
-- 効力条件  # なし
--
eq nw(mes1(S,P1,P2,PA)) = m1(P1,P1,P2,PA,comp(PA,rand(P1))) , nw(S) .


-- for mes2
--
-- mes2 : P2 は P1 に自分を認証してもらうため、message2 を送信
--
-- 効力条件  # 以下の条件を満たす message1 が存在
--             1. 受信者が P2
--             2. 送信者が P1
--
op c-mes2 : System Principal Principal Msg -> Bool
eq c-mes2(S,P2,P1,M) =
M \in nw(S) and m1?(M) and receiver(M) = P2 and sender(M) = P1 .
--
```

```
ceq nw(mes2(S,P2,P1,M)) =
m2(P2,P1,comp(pa(M),rand(P2)),cert(P2,pa(M)),
   enc(skey(co(M),rand(P2)),sign(P2,comp(pa(M),rand(P2))))) , nw(S)
if c-mes2(S,P2,P1,M) .
ceq mes2(S,P2,P1,M) = S if not c-mes2(S,P2,P1,M) .


-- for mes3
--
-- mes3 : P1 は P2 の認証後、自分を認証してもらうため、P2 に message3 を送信
--
-- 効力条件  # 以下の条件を満たす message1(M1) が存在
--             1. 作成者が P1
--             2. 受信者が P2
--           # 以下の条件を満たす message2(M2) が存在
--             1. 証明書内の識別子 (公開鍵) は P2
--             2. 証明書内のパラメータが M1 のものと一致
--             3. P1 は暗号文を解読可能
--             4. 暗号文内の署名が P2
--             5. 署名内の構成要素が平文のものと一致
--
op c-mes3 : System Principal Principal Msg Msg -> Bool
eq c-mes3(S,P1,P2,M1,M2) =
M1 \in nw(S) and m1?(M1) and
creator(M1) = P1 and receiver(M1) = P2 and
M2 \in nw(S) and m2?(M2) and
pr(ce(M2)) = P2 and
pa(ce(M2)) = pa(M1) and
co(M1) = com(skey(ci(M2))) and
pr(sign(ci(M2))) = P2 and
co(sign(ci(M2))) = co(M2) .
--
ceq nw(mes3(S,P1,P2,M1,M2)) =
m3(P1,P2,cert(P1,pa(M1)),
   enc(skey(co(M2),rand(co(M1))),sign(P1,co(M1)))) , nw(S)
if c-mes3(S,P1,P2,M1,M2) .
ceq mes3(S,P1,P2,M1,M2) = S if not c-mes3(S,P1,P2,M1,M2) .


-- for mes4
--
-- mes4 : 侵入者ではない P1 は P2 の認証後、秘密情報を
--        含んだ暗号文を P2 に送る
--
-- 効力条件  # P1 は侵入者ではない
--           # c-mes3
--
op c-mes4 : System Principal Principal Msg Msg -> Bool
eq c-mes4(S,P1,P2,M1,M2) = not(P1 = intruder) and c-mes3(S,P1,P2,M1,M2) .
--
ceq nw(mes4(S,P1,P2,M1,M2)) =
m4(P1,enc(P2,skey(co(M2),rand(co(M1))))) ,
nw(S) if c-mes4(S,P1,P2,M1,M2) .
ceq mes4(S,P1,P2,M1,M2) = S if not c-mes4(S,P1,P2,M1,M2) .


-- for mes5
--
-- mes5 : 侵入者ではない P2 は P1 の認証後、秘密情報を
--        含んだ暗号文を P1 に送る
--
-- 効力条件  # P2 は侵入者ではない
--           # 以下の条件を満たす message2(M2) が存在
--             1. 作成者が P2
--             2. 受信者が P1
--           # 以下の条件を満たす message3(M3) が存在
```

```
--              1. 証明書内の識別子 (公開鍵) は P1
--              2. 証明書内のパラメータが M2 の証明書内のものと一致
--              3. P2 が暗号文を解読可能
--              4. 暗号文内の署名が P1
--              5. 署名内の構成要素が M2 の鍵のものと一致
--
op c-mes5 : System Principal Principal Msg Msg -> Bool
eq c-mes5(S,P2,P1,M2,M3) = not(P2 = intruder) and
M2 \in nw(S) and m2?(M2) and
creator(M2) = P2 and receiver(M2) = P1 and
M3 \in nw(S) and m3?(M3) and
pr(ce(M3)) = P1 and
pa(ce(M3)) = pa(ce(M2)) and
co(M2) = com(skey(ci(M3))) and
pr(sign(ci(M3))) = P1 and
co(sign(ci(M3))) = com(skey(ci(M2))) .
--
ceq nw(mes5(S,P2,P1,M2,M3)) =
m5(P2,enc(P1,skey(ci(M2)))) , nw(S) if c-mes5(S,P2,P1,M2,M3) .
ceq mes5(S,P2,P1,M2,M3) = S if not c-mes5(S,P2,P1,M2,M3) .


-- faking message
-- for message1
bop fkm11 : System Principal Principal Parameter Parameter -> System
bop fkm12 : System Principal Principal Parameter Component -> System

** message1 の偽造に必要な構成要素

-- for fkm11
-- 構成要素を侵入者が作成

-- for fkm12
-- 入手した構成要素を利用
op c-fkm12 : System Component -> Bool
eq c-fkm12(S,CO) = CO \in cco(nw(S)) .


-- for fkm11
eq nw(fkm11(S,P1,P2,PA1,PA2)) =
m1(intruder,P1,P2,PA1,comp(PA2,rand(intruder))) , nw(S) .

-- for fkm12
ceq nw(fkm12(S,P1,P2,PA,CO)) = m1(intruder,P1,P2,PA,CO) , nw(S)
if c-fkm12(S,CO) .
ceq fkm12(S,P1,P2,PA,CO) = S if not c-fkm12(S,CO) .


-- for message2
bop fkm211 : System Principal Parameter Certificate Component Sign -> System
bop fkm212 : System Principal Component Certificate Component Sign -> System
bop fkm221 : System Principal Parameter Certificate Cipher -> System
bop fkm222 : System Principal Component Certificate Cipher -> System

** message2 の偽造に必要な暗号文

-- for fkm21
-- 入手した構成要素および署名を利用
op c-fkm21 : System Component Sign -> Bool
eq c-fkm21(S,CO,SI) = CO \in cco(nw(S)) and SI \in csi(nw(S)) .

-- for fkm22
-- 入手した暗号文を利用
op c-fkm22 : System Cipher -> Bool
eq c-fkm22(S,CI) = CI \in cci(nw(S)) .
```

```
** message2 の偽造に必要な構成要素

-- for fkm201
-- 構成要素を侵入者が作成

-- for fkm202
-- 入手した構成要素を利用
op c-fkm202 : System Component -> Bool
eq c-fkm202(S,CO) = CO \in cco(nw(S)) .


-- for fkm211
-- # fkm21 and fkm201
op c-fkm211 : System Component Sign -> Bool
eq c-fkm211(S,CO,SI) = c-fkm21(S,CO,SI) .
--
ceq nw(fkm211(S,P,PA,CE,CO,SI)) =
m2(intruder,P,comp(PA,rand(intruder)),CE,
   enc(skey(CO,rand(intruder)),SI)) , nw(S) if c-fkm211(S,CO,SI) .
ceq fkm211(S,P,PA,CE,CO,SI) = S if not c-fkm211(S,CO,SI) .

-- for fkm212
-- # fkm21 and fkm202
op c-fkm212 : System Component Component Sign -> Bool
eq c-fkm212(S,CO1,CO2,SI) = c-fkm21(S,CO1,SI) and c-fkm202(S,CO2) .
--
ceq nw(fkm212(S,P,CO1,CE,CO2,SI)) =
m2(intruder,P,CO1,CE,
   enc(skey(CO2,rand(intruder)),SI)) , nw(S) if c-fkm212(S,CO1,CO2,SI) .
ceq fkm212(S,P,CO1,CE,CO2,SI) = S if not c-fkm212(S,CO1,CO2,SI) .

-- for fkm221
-- # fkm22 and fkm201
op c-fkm221 : System Cipher -> Bool
eq c-fkm221(S,CI) = c-fkm22(S,CI) .
--
ceq nw(fkm221(S,P,PA,CE,CI)) =
m2(intruder,P,comp(PA,rand(intruder)),CE,CI) , nw(S) if c-fkm221(S,CI) .
ceq fkm221(S,P,PA,CE,CI) = S if not c-fkm221(S,CI) .

-- for fkm222
-- # fkm22 and fkm202
op c-fkm222 : System Component Cipher -> Bool
eq c-fkm222(S,CO,CI) = c-fkm22(S,CI) and c-fkm202(S,CO) .
--
ceq nw(fkm222(S,P,CO,CE,CI)) =
m2(intruder,P,CO,CE,CI) , nw(S) if c-fkm222(S,CO,CI) .
ceq fkm222(S,P,CO,CE,CI) = S if not c-fkm222(S,CO,CI) .


-- for message3
bop fkm31 : System Principal Certificate Component Sign -> System
bop fkm32 : System Principal Certificate Cipher -> System

** message3 の偽造に必要な暗号文について

-- for fkm31
-- 入手した構成要素および署名を利用
op c-fkm31 : System Component Sign -> Bool
eq c-fkm31(S,CO,SI) = c-fkm21(S,CO,SI) .

-- for fkm32
-- 入手した暗号文を利用
op c-fkm32 : System Cipher -> Bool
eq c-fkm32(S,CI) = c-fkm22(S,CI) .
```

```
-- for fkm31
--
ceq nw(fkm31(S,P,CE,CO,SI)) =
m3(intruder,P,CE,
   enc(skey(CO,rand(intruder)),SI)) , nw(S) if c-fkm31(S,CO,SI) .
ceq fkm31(S,P,CE,CO,SI) = S if not c-fkm31(S,CO,SI) .

-- for fkm32
--
ceq nw(fkm32(S,P,CE,CI)) =
m3(intruder,P,CE,CI) , nw(S)
if c-fkm32(S,CI) .
ceq fkm32(S,P,CE,CI) = S if not c-fkm32(S,CI) .
}
```

付 録B　検証したい性質の記述 (CafeOBJ)

```
-- ----------------------------------------------------------------
-- Invariant
-- Begin Secrecy Theorem
--
-- ----------------------------------------------------------------

mod INV {
pr(STS)
--
op si : -> Sign
op ci : -> Cipher
op ms : -> Msg
op cm : -> CipherM

-- declare invariants to prove
op inv-10 : System Msg -> Bool
op inv000 : System Sign -> Bool
op inv010 : System Cipher -> Bool
op inv020 : System Msg -> Bool
op inv030 : System Msg -> Bool
op inv100 : System CipherM -> Bool
op inv200 : System Msg -> Bool
op inv300 : System Msg -> Bool

-- CafeOBJ variables to prove
var S : System
var SI : Sign
var CI : Cipher
var MS : Msg
var CM : CipherM

-- define invariants to prove
eq inv-10(S,MS) = MS \in nw(S) and m1?(MS) and not(creator(MS) = intruder)
                  implies pr(rand(co(MS))) = creator(MS) .
eq inv000(S,SI) = SI \in csi(nw(S)) implies pr(rand(co(SI))) = pr(SI) .
eq inv010(S,CI) = CI \in cci(nw(S)) implies pr(rand(co(sign(CI)))) = pr(sign(CI)) .
eq inv020(S,MS) = MS \in nw(S) and m2?(MS) and not(pr(sign(ci(MS))) = intruder)
                  implies pr(rand(co(sign(ci(MS))))) = pr(sign(ci(MS))) .
eq inv030(S,MS) = MS \in nw(S) and m3?(MS) and not(pr(sign(ci(MS))) = intruder)
                  implies pr(rand(co(sign(ci(MS))))) = pr(sign(ci(MS))) .
eq inv100(S,CM) = CM \in ccm(nw(S)) implies (for(CM) = intruder) .
eq inv200(S,MS) = MS \in nw(S) and m4?(MS) and
                  not(creator(MS) = intruder) and
                  not(for(cm(MS)) = intruder) implies
                  for(cm(MS)) = pr(rand(com(key(cm(MS))))) .
eq inv300(S,MS) = MS \in nw(S) and m5?(MS) and
                  not(creator(MS) = intruder) and
                  not(for(cm(MS)) = intruder) implies
                  for(cm(MS)) = pr(rand(com(key(cm(MS))))) .
}
```

```
mod ISTEP {
pr(INV)
-- arbitrary objects
ops s s' : -> System

-- declare predicates to prove in induction step
op istep-10 : Msg -> Bool
op istep000 : Sign -> Bool
op istep010 : Cipher -> Bool
op istep020 : Msg -> Bool
op istep030 : Msg -> Bool
op istep100 : CipherM -> Bool
op istep200 : Msg -> Bool
op istep300 : Msg -> Bool

-- CafeOBJ variables
var SI : Sign
var CI : Cipher
var MS : Msg
var CM : CipherM

-- define predicates to prove in induction step
eq istep-10(MS) = inv-10(s,MS) implies inv-10(s',MS) .
eq istep000(SI) = inv000(s,SI) implies inv000(s',SI) .
eq istep010(CI) = inv010(s,CI) implies inv010(s',CI) .
eq istep020(MS) = inv020(s,MS) implies inv020(s',MS) .
eq istep030(MS) = inv030(s,MS) implies inv030(s',MS) .
eq istep100(CM) = inv100(s,CM) implies inv100(s',CM) .
eq istep200(MS) = inv200(s,MS) implies inv200(s',MS) .
eq istep300(MS) = inv300(s,MS) implies inv300(s',MS) .
}
```

付 録C　表明1の証明譜(CafeOBJ)

```
--> inv100

--> ******************************************************
-- I) Base Case

open INV .
red inv100(init,cm) .
close .


-- II) Inductive case
--> ******************************************************
--> 1) mes1(s,p10,p20,pa10)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
op pa10 : -> Parameter .
-- successor state
eq s' = mes1(s,p10,p20,pa10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 2) mes2(s,p20,p10,m10)
-- 2.1) c-mes2(s,p20,p10,m10)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
op m10 : -> Msg .
-- assumption
eq c-mes2(s,p20,p10,m10) = true .
-- successor state
eq s' = mes2(s,p20,p10,m10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 2.2) not c-mes2(s,p20,p10,m10)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
op m10 : -> Msg .
-- assumption
eq c-mes2(s,p20,p10,m10) = false .
-- successor state
eq s' = mes2(s,p20,p10,m10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 3) mes3(s,p10,p20,m10,m20)
```

```
-- 3.1) c-mes3(s,p10,p20,m10,m20)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m10 m20 : -> Msg .
-- assumption
eq c-mes3(s,p10,p20,m10,m20) = true .
-- successor state
eq s' = mes3(s,p10,p20,m10,m20) .
-- check if the principal is true
red istep100(cm) .
close .


-- 3.2) not c-mes3(s,p10,p20,m10,m20)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m10 m20 : -> Msg .
-- assumption
eq c-mes3(s,p10,p20,m10,m20) = false .
-- successor state
eq s' = mes3(s,p10,p20,m10,m20) .
-- check if the principal is true
red istep100(cm) .
close .


--> 4) mes4(s,p10,p20,m10,m20)
-- 4.1) c-mes4(s,p10,p20,m10,m20)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m10 m20 : -> Msg .
-- assumption
eq c-mes4(s,p10,p20,m10,m20) = true .
--
eq (cm = enc(p20,skey(co(m20),rand(co(m10))))) = false .
-- successor state
eq s' = mes4(s,p10,p20,m10,m20) .
-- check if the principal is true
red istep100(cm) .
close .

open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m10 m20 : -> Msg .
-- assumption
eq c-mes4(s,p10,p20,m10,m20) = true .
--
eq cm = enc(p20,skey(co(m20),rand(co(m10)))) .
--
eq rsk(skey(co(m20),rand(co(m10)))) = false .
-- successor state
eq s' = mes4(s,p10,p20,m10,m20) .
-- check if the principal is true
red istep100(cm) .
close .

open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m10 m20 : -> Msg .
-- assumption
eq c-mes4(s,p10,p20,m10,m20) = true .
```

```
--
eq cm = enc(p20,skey(co(m20),rand(co(m10)))) .
--
-- eq rsk(skey(co(m20),rand(co(m10)))) = true .
eq pr(rand(co(m20))) = intruder .
--
-- from c-mes4(s,p10,p20,m10,m20) = true .
eq m20 \in nw(s) and m2?(m20) = true .
eq pr(sign(ci(m20))) = p20 .
eq co(sign(ci(m20))) = co(m20) .
--
-- set hypothesis
op h10 : -> Bool .
eq h10 = (pr(rand(co(sign(ci(m20))))) = pr(sign(ci(m20)))) .
-- red pr(rand(co(sign(ci(m20))))) .
-- ** result : intruder
-- red pr(sign(ci(m20))) .
-- ** result : p20
-- if h10 then (p20 = intruder)
-- red inv020(s,m20) implies h10 .
-- ** result : true
-- -->
eq p20 = intruder .
-- successor state
eq s' = mes4(s,p10,p20,m10,m20) .
-- check if the principal is true
red istep100(cm) .
close .


-- 4.2) not c-mes3(s,p10,p20,m10,m20)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m10 m20 : -> Msg .
-- assumption
eq c-mes4(s,p10,p20,m10,m20) = false .
-- successor state
eq s' = mes4(s,p10,p20,m10,m20) .
-- check if the principal is true
red istep100(cm) .
close .


--> 5) mes5(s,p20,p10,m20,m30)
-- 5.1) c-mes5(s,p20,p30,m20,m30)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m20 m30 : -> Msg .
-- assumption
eq c-mes5(s,p20,p10,m20,m30) = true .
--
eq (cm = enc(p10,skey(ci(m20)))) = false .
-- successor state
eq s' = mes5(s,p20,p10,m20,m30) .
-- check if the principal is true
red istep100(cm) .
close .

open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m20 m30 : -> Msg .
-- assumption
eq c-mes5(s,p20,p10,m20,m30) = true .
```

```
--
eq cm = enc(p10,skey(ci(m20))) .
--
eq rsk(skey(ci(m20))) = false .
-- successor state
eq s' = mes5(s,p20,p10,m20,m30) .
-- check if the principal is true
red istep100(cm) .
close .

open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m20 m30 : -> Msg .
-- assumption
eq c-mes5(s,p20,p10,m20,m30) = true .
--
eq cm = enc(p10,skey(ci(m20))) .
--
-- eq rsk(skey(ci(m20))) = true .
eq pr(rand(com(skey(ci(m20))))) = intruder .
--
-- from c-mes5(s,p20,p10,m20,m30) = true .
eq m30 \in nw(s) and m3?(m30) = true .
eq pr(sign(ci(m30))) = p10 .
eq co(sign(ci(m30))) = com(skey(ci(m20))) .
--
-- set hypothesis
op h10 : -> Bool .
eq h10 = (pr(rand(co(sign(ci(m30))))) = pr(sign(ci(m30)))) .
-- red pr(rand(co(sign(ci(m30))))) .
-- ** result : intruder
-- red pr(sign(ci(m30))) .
-- ** result : p10
-- if h10 then (p10 = intruder)
-- red inv030(s,m30) implies h10 .
-- ** result : true
-- -->
eq p10 = intruder .
-- successor state
eq s' = mes5(s,p20,p10,m20,m30) .
-- check if the principal is true
red istep100(cm) .
close .


-- 5.2) not c-mes5(s,p20,p10,m20,m30)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops m20 m30 : -> Msg .
-- assumption
eq c-mes5(s,p20,p10,m20,m30) = false .
-- successor state
eq s' = mes5(s,p20,p10,m20,m30) .
-- check if the principal is true
red istep100(cm) .
close .


--> 6) fkm11(s,p10,p20,pa10,pa20)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
ops pa10 pa20 : -> Parameter .
-- successor state
```

```
eq s' = fkm11(s,p10,p20,pa10,pa20) .
-- check if the principal is true
red istep100(cm) .
close .


--> 7) fkm12(s,p10,p20,pa10,co10)
-- 7.1) c-fkm12(s,co10)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
op pa10 : -> Parameter .
op co10 : -> Component .
-- assumption
eq c-fkm12(s,co10) = true .
-- successor state
eq s' = fkm12(s,p10,p20,pa10,co10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 7.2) not c-fkm12(s,co10)
open ISTEP .
-- arbitrary object
ops p10 p20 : -> Principal .
op pa10 : -> Parameter .
op co10 : -> Component .
-- assumption
eq c-fkm12(s,co10) = false .
-- successor state
eq s' = fkm12(s,p10,p20,pa10,co10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 8) fkm211(s,p10,pa10,ce10,co10,si10)
-- 8.1) c-fkm211(s,co10,si10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op pa10 : -> Parameter .
op ce10 : -> Certificate .
op co10 : -> Component .
op si10 : -> Sign .
-- assumption
eq c-fkm211(s,co10,si10) = true .
-- successor state
eq s' = fkm211(s,p10,pa10,ce10,co10,si10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 8.2) not c-fkm211(s,m10,co10,si10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op pa10 : -> Parameter .
op ce10 : -> Certificate .
op co10 : -> Component .
op si10 : -> Sign .
-- assumption
eq c-fkm211(s,co10,si10) = false .
-- successor state
```

```
eq s' = fkm211(s,p10,pa10,ce10,co10,si10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 9) fkm212(s,p10,co10,ce10,co20,si10)
-- 9.1) c-fkm212(s,co10,co20,si10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op ce10 : -> Certificate .
op si10 : -> Sign .
ops co10 co20 : -> Component .
-- assumption
eq c-fkm212(s,co10,co20,si10) = true .
-- successor state
eq s' = fkm212(s,p10,co10,ce10,co20,si10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 9.2) not c-fkm212(s,co10,co20,si10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op ce10 : -> Certificate .
op si10 : -> Sign .
ops co10 co20 : -> Component .
-- assumption
eq c-fkm212(s,co10,co20,si10) = false .
-- successor state
eq s' = fkm212(s,p10,co10,ce10,co20,si10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 10) fkm221(s,p10,pa10,ce10,ci10)
-- 10.1) c-fkm221(s,ci10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op pa10 : -> Parameter .
op ce10 : -> Certificate .
op ci10 : -> Cipher .
-- assumption
eq c-fkm221(s,ci10) = true .
-- successor state
eq s' = fkm221(s,p10,pa10,ce10,ci10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 10.2) not c-fkm221(s,m10,ci10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op pa10 : -> Parameter .
op ce10 : -> Certificate .
op ci10 : -> Cipher .
-- assumption
eq c-fkm221(s,ci10) = false .
-- successor state
```

```
eq s' = fkm221(s,p10,pa10,ce10,ci10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 11) fkm222(s,p10,co10,ce10,ci10)
-- 11.1) c-fkm222(s,co10,ci10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op co10 : -> Component .
op ce10 : -> Certificate .
op ci10 : -> Cipher .
-- assumption
eq c-fkm222(s,co10,ci10) = true .
-- successor state
eq s' = fkm222(s,p10,co10,ce10,ci10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 11.2) not c-fkm222(s,co10,ci10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op co10 : -> Component .
op ce10 : -> Certificate .
op ci10 : -> Cipher .
-- assumption
eq c-fkm222(s,co10,ci10) = false .
-- successor state
eq s' = fkm222(s,p10,co10,ce10,ci10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 12) fkm31(s,p10,ce10,co10,si10)
-- 12.1) c-fkm310(s,co10,si10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op ce10 : -> Certificate .
op co10 : -> Component .
op si10 : -> Sign .
-- assumption
eq c-fkm31(s,co10,si10) = true .
-- successor state
eq s' = fkm31(s,p10,ce10,co10,si10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 12.2) not c-fkm31(s,co10,si10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op ce10 : -> Certificate .
op co10 : -> Component .
op si10 : -> Sign .
-- assumption
eq c-fkm31(s,co10,si10) = false .
-- successor state
```

```
eq s' = fkm31(s,p10,ce10,co10,si10) .
-- check if the principal is true
red istep100(cm) .
close .


--> 13) fkm32(s,p10,ce10,ci10)
-- 13.1) c-fkm32(s,ci10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op ce10 : -> Certificate .
op ci10 : -> Cipher .
-- assumption
eq c-fkm32(s,ci10) = true .
-- successor state
eq s' = fkm32(s,p10,ce10,ci10) .
-- check if the principal is true
red istep100(cm) .
close .


-- 13.2) not c-fkm32(s,ci10)
open ISTEP .
-- arbitrary object
op p10 : -> Principal .
op ce10 : -> Certificate .
op ci10 : -> Cipher .
-- assumption
eq c-fkm32(s,ci10) = false .
-- successor state
eq s' = fkm32(s,p10,ce10,ci10) .
-- check if the principal is true
red istep100(cm) .
close
```

付 録D　STSプロトコルの仕様(Coq)

```
(* STS protocol *)
Require PolyList.


(* 主体の定義 *)

Inductive Principal : Set :=
  intruder : Principal
| P : nat -> Principal.


(* パラメータの定義 *)

Inductive Param : Set :=
  PA : nat -> Param .


(* 乱数の定義 *)

Inductive Randum : Set :=
  RAND : Principal -> Randum.


(* 構成要素の定義 *)

Section Components.

Inductive Component : Set :=
  COMP : Param -> Randum -> Component.

Definition whoz_com [CO:Component] :=
  Cases CO of (COMP _ (RAND p)) => p end.

End Components.


(* セッション鍵の定義 *)

Section Session.

Inductive Skey : Set :=
  SKEY : Component -> Randum -> Skey.

Definition key_for [SK:Skey] :=
  Cases SK of (SKEY co _) => (whoz_com co) end.

Definition rsk [SK:Skey] : Prop := (key_for SK) = intruder.

End Session.


(* 証明書の定義 *)

Inductive Certificate : Set :=
```

```
  CERT : Principal -> Param -> Certificate.


(* 署名の定義 *)

Inductive Sign : Set :=
  SIGN : Principal -> Component -> Sign.


(* 暗号文の定義 *)

Section Ciphers.

Inductive Cipher : Set :=
  ENC : Skey -> Sign -> Cipher.

Definition skey [C:Cipher] :=
  Cases C of (ENC x _) => x end.

End Ciphers.


(* 通信暗号文の定義 *)

Section CipherMs.

Inductive CipherM : Set :=
  ENCM : Principal -> Skey -> CipherM.

Definition for [CM:CipherM] :=
  Cases CM of (ENCM x _) => x end.

Definition skey2 [CM:CipherM] :=
  Cases CM of (ENCM _ y) => y end.

End CipherMs.


(* メッセージの定義 *)

Inductive Msg : Set :=
  M1 : Principal -> Principal -> Principal ->
       Param -> Component -> Msg
| M2 : Principal -> Principal -> Component ->
       Certificate -> Cipher -> Msg
| M3 : Principal -> Principal -> Certificate -> Cipher -> Msg
| M4 : Principal -> CipherM -> Msg
| M5 : Principal -> CipherM -> Msg.


(* ネットワークの定義 *)

Section networks.

Definition Network := (list Msg).
Definition ColComp := (list Component).
Definition ColSign := (list Sign).
Definition ColCipher := (list Cipher).
Definition ColCipherM := (list CipherM).

Inductive cco : Component -> Network -> Prop :=
  cco1 : (co:Component;n:Network;m:nat;p1,p2:Principal;pa:Param)
         (cco co (cons (M1 (P m) p1 p2 pa co) n))
| cco2 : (co1,co2:Component;n:Network;p1,p2,p3:Principal;pa:Param)
         (cco co1 n) -> ~co1=co2
       -> (cco co1 (cons (M1 p1 p2 p3 pa co2) n))
```

```
| cco3 : (co:Component;n:Network;m:nat;p:Principal;ce:Certificate;c:Cipher)
          (cco co (cons (M2 (P m) p co ce c) n))
| cco4 : (co1,co2:Component;n:Network;p1,p2:Principal;ce:Certificate;c:Cipher)
          (cco co1 n) -> ~co1=co2
       -> (cco co1 (cons (M2 p1 p2 co2 ce c) n))
| cco5 : (co:Component;n:Network;p1,p2:Principal;ce:Certificate;c:Cipher)
          (cco co n)
       -> (cco co (cons (M3 p1 p2 ce c) n))
| cco6 : (co:Component;n:Network;p:Principal;cm:CipherM)
          (cco co n)
       -> (cco co (cons (M4 p cm) n))
| cco7 : (co:Component;n:Network;p:Principal;cm:CipherM)
          (cco co n)
       -> (cco co (cons (M5 p cm) n)).


Inductive csi : Sign -> Network -> Prop :=
  csi1 : (si:Sign;n:Network;m:nat;p1,p2,p3:Principal;pa:Param;co:Component)
          (csi si n)
       -> (csi si (cons (M1 p1 p2 p2 pa co) n))
| csi2 : (si:Sign;n:Network;m:nat;p:Principal;co:Component;ce:Certificate)
         (sk:Skey)
          (rsk sk)
       -> (csi si (cons (M2 (P m) p co ce (ENC sk si)) n))
| csi3 : (si1,si2:Sign;n:Network;m:nat;p1,p2:Principal;co:Component)
         (ce:Certificate;sk:Skey)
          (csi si1 n) -> ~si1=si2
       -> (csi si1 (cons (M2 p1 p2 co ce (ENC sk si2)) n))
| csi4 : (si:Sign;n:Network;m:nat;p:Principal;ce:Certificate;sk:Skey)
          (rsk sk)
       -> (csi si (cons (M3 (P m) p ce (ENC sk si)) n))
| csi5 : (si1,si2:Sign;n:Network;m:nat;p1,p2:Principal;ce:Certificate)
         (sk:Skey)
          (csi si1 n) -> ~si1=si2
       -> (csi si1 (cons (M3 p1 p2 ce (ENC sk si2)) n))
| csi6 : (si:Sign;n:Network;p:Principal;cm:CipherM)
          (csi si n)
       -> (csi si (cons (M4 p cm) n))
| csi7 : (si:Sign;n:Network;p:Principal;cm:CipherM)
          (csi si n)
       -> (csi si (cons (M5 p cm) n)).


Inductive cci : Cipher -> Network -> Prop :=
  cci1 : (c:Cipher;n:Network;m:nat;p1,p2,p3:Principal;pa:Param;co:Component)
          (cci c n)
       -> (cci c (cons (M1 p1 p2 p2 pa co) n))
| cci2 : (c:Cipher;n:Network;m:nat;p:Principal;co:Component;ce:Certificate)
          ~(rsk (skey c))
       -> (cci c (cons (M2 (P m) p co ce c) n))
| cci3 : (c1,c2:Cipher;n:Network;m:nat;p1,p2:Principal;co:Component)
         (ce:Certificate)
          (cci c1 n) -> ~c1=c2
       -> (cci c1 (cons (M2 p1 p2 co ce c2) n))
| cci4 : (c:Cipher;n:Network;m:nat;p:Principal;ce:Certificate)
          ~(rsk (skey c))
       -> (cci c (cons (M3 (P m) p ce c) n))
| cci5 : (c1,c2:Cipher;n:Network;m:nat;p1,p2:Principal;ce:Certificate)
          (cci c1 n) -> ~c1=c2
       -> (cci c1 (cons (M3 p1 p2 ce c2) n))
| cci6 : (c:Cipher;n:Network;p:Principal;cm:CipherM)
          (cci c n)
       -> (cci c (cons (M4 p cm) n))
| cci7 : (c:Cipher;n:Network;p:Principal;cm:CipherM)
          (cci c n)
       -> (cci c (cons (M5 p cm) n)).
```

```
Inductive ccm : CipherM -> Network -> Prop :=
  ccm1 : (cm:CipherM;n:Network;m:nat;p1,p2,p3:Principal;pa:Param;co:Component)
          (ccm cm n)
       -> (ccm cm (cons (M1 p1 p2 p2 pa co) n))
| ccm2 : (cm:CipherM;n:Network;m:nat;p1,p2:Principal;co:Component)
         (ce:Certificate;c:Cipher)
          (ccm cm n)
       -> (ccm cm (cons (M2 p1 p2 co ce c) n))
| ccm3 : (cm:CipherM;n:Network;m:nat;p1,p2:Principal;ce:Certificate;c:Cipher)
          (ccm cm n)
       -> (ccm cm (cons (M3 p1 p2 ce c) n))
| ccm4 : (cm:CipherM;n:Network;p:Principal)
          (rsk (skey2 cm))
       -> (ccm cm (cons (M4 p cm) n))
| ccm5 : (cm1,cm2:CipherM;n:Network;p:Principal)
          (ccm cm1 n) -> ~cm1=cm2
       -> (ccm cm1 (cons (M4 p cm2) n))
| ccm6 : (cm:CipherM;n:Network;p:Principal)
          (rsk (skey2 cm))
       -> (ccm cm (cons (M5 p cm) n))
| ccm7 : (cm1,cm2:CipherM;n:Network;p:Principal)
          (ccm cm1 n) -> ~cm1=cm2
       -> (ccm cm1 (cons (M5 p cm2) n)).

End networks.


(* STS *)

Section STSs.

Inductive System : Set :=
  init : System
| mes1 : System -> Principal -> Principal -> Param -> System
| mes2 : System -> Principal -> Principal -> Param -> Component ->
         System
| mes3 : System -> Principal -> Principal -> Param -> Component ->
         Component -> System
| mes4 : System -> nat -> Principal -> Param -> Component ->
         Component -> System
| mes5 : System -> nat -> Principal -> Param -> Component ->
         Component -> System
| fkm11 : System -> Principal -> Principal -> Param -> Param -> System
| fkm12 : System -> Principal -> Principal -> Param -> Component ->
          System
| fkm211 : System -> Principal -> Param -> Certificate -> Component ->
           Sign -> System
| fkm212 : System -> Principal -> Component -> Certificate -> Component ->
           Sign -> System
| fkm221 : System -> Principal -> Param -> Certificate -> Cipher ->
           System
| fkm222 :  System -> Principal -> Component -> Certificate -> Cipher ->
            System
| fkm31 : System -> Principal -> Certificate -> Component -> Sign ->
          System
| fkm32 : System -> Principal -> Certificate -> Cipher -> System.

Definition cmes2
[n:Network;p1,p2:Principal;pa:Param;co:Component] : Prop :=
(EX Q:Principal | (In (M1 Q p1 p2 pa co) n)).

Definition cmes3
[n:Network;p1,p2:Principal;pa:Param;co1,co2:Component] : Prop :=
(In (M1 p1 p1 p2 pa co1) n) /\
```

```
(EX Q:Principal | (EX R:Randum |
(In (M2 Q p1 co2 (CERT p2 pa) (ENC (SKEY co1 R) (SIGN p2 co2))) n))).

Definition cmes4
[n:Network;m:nat;p:Principal;pa:Param;co1,co2:Component] : Prop :=
(In (M1 (P m) (P m) p pa co1) n) /\
(EX Q:Principal | (EX R:Randum |
(In (M2 Q (P m) co2 (CERT p pa) (ENC (SKEY co1 R) (SIGN p co2))) n))).

Definition cmes5
[n:Network;m:nat;p:Principal;pa:Param;co1,co2:Component] : Prop :=
(In (M2 (P m) p co1 (CERT (P m) pa)
        (ENC (SKEY co2 (RAND (P m))) (SIGN (P m) co1))) n) /\
(EX Q:Principal | (EX R:Randum |
(In (M3 Q (P m) (CERT p pa) (ENC (SKEY co1 R) (SIGN p co2))) n))).

Definition cfkm12 [n:Network;co:Component] : Prop := (cco co n).

Definition cfkm211 [n:Network;co:Component;si:Sign] : Prop :=
(cco co n) /\ (csi si n).

Definition cfkm212
[n:Network;co1,co2:Component;si:Sign] : Prop :=
(cco co1 n) /\ (cco co2 n) /\ (csi si n).

Definition cfkm221 [n:Network;c:Cipher] : Prop := (cci c n).

Definition cfkm222 [n:Network;co:Component;c:Cipher] : Prop :=
(cco co n) /\ (cci c n).

Definition cfkm31 [n:Network;co:Component;si:Sign] : Prop :=
(cco co n) /\ (csi si n).

Definition cfkm32 [n:Network;c:Cipher] : Prop := (cci c n).


Inductive nw : System -> Network -> Prop :=
  nw_init : (nw init (nil Msg))
| nw_mes1 : (s:System;n:Network;p1,p2:Principal;pa:Param)
             (nw s n)
          -> (nw (mes1 s p1 p2 pa)
             (cons (M1 p1 p1 p2 pa (COMP pa (RAND p1))) n))
| nw_mes2a : (s:System;n:Network;p1,p2:Principal;pa:Param;co:Component)
             (nw s n) -> (cmes2 n p1 p2 pa co) ->
             (nw (mes2 s p1 p2 pa co)
             (cons (M2 p2 p1 (COMP pa (RAND p2)) (CERT p2 pa)
                             (ENC (SKEY co (RAND p2))
                             (SIGN p2 (COMP pa (RAND p2))))) n))
| nw_mes2b : (s:System;n:Network;p1,p2:Principal;pa:Param;co:Component)
             (nw s n) -> ~(cmes2 n p1 p2 pa co) ->
             (nw (mes2 s p1 p2 pa co) n)
| nw_mes3a : (s:System;n:Network;p1,p2:Principal)
             (pa:Param;co1,co2:Component)
             (nw s n) -> (cmes3 n p1 p2 pa co1 co2) ->
             (nw (mes3 s p1 p2 pa co1 co2)
             (cons (M3 p1 p2 (CERT p1 pa)
             (ENC (SKEY co2 (RAND p1)) (SIGN p1 co1))) n))
| nw_mes3b : (s:System;n:Network;p1,p2:Principal)
             (pa:Param;co1,co2:Component)
             (nw s n) -> ~(cmes3 n p1 p2 pa co1 co2) ->
             (nw (mes3 s p1 p2 pa co1 co2) n)
| nw_mes4a : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;co1,co2:Component)
             (nw s n) -> (cmes4 n m p pa co1 co2) ->
             (nw (mes4 s m p pa co1 co2)
             (cons (M4 (P m) (ENCM p (SKEY co2 (RAND (P m))))) n))
```

```
| nw_mes4b : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;co1,co2:Component)
             (nw s n) -> ~(cmes4 n m p pa co1 co2) ->
             (nw (mes4 s m p pa co1 co2) n)
| nw_mes5a : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;co1,co2:Component)
             (nw s n) -> (cmes5 n m p pa co1 co2) ->
             (nw (mes5 s m p pa co1 co2)
             (cons (M5 (P m) (ENCM p (SKEY co2 (RAND (P m))))) n))
| nw_mes5b : (s:System;n:Network;m:nat;p:Principal)
             (pa:Param;co1,co2:Component)
             (nw s n) -> ~(cmes5 n m p pa co1 co2) ->
             (nw (mes5 s m p pa co1 co2) n)
| nw_fkm11 : (s:System;n:Network;p1,p2:Principal;pa1,pa2:Param)
             (nw s n)
          -> (nw (fkm11 s p1 p2 pa1 pa2)
             (cons (M1 intruder p1 p2 pa1 (COMP pa2 (RAND intruder))) n))
| nw_fkm12a : (s:System;n:Network;p1,p2:Principal;pa:Param;co:Component)
             (nw s n) -> (cfkm12 n co) ->
             (nw (fkm12 s p1 p2 pa co)
             (cons (M1 intruder p1 p2 pa co) n))
| nw_fkm12b : (s:System;n:Network;p1,p2:Principal;pa:Param;co:Component)
             (nw s n) -> ~(cfkm12 n co) ->
             (nw (fkm12 s p1 p2 pa co) n)
| nw_fkm211a : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (co:Component;si:Sign)
             (nw s n) -> (cfkm211 n co si) ->
             (nw (fkm211 s p pa ce co si)
             (cons (M2 intruder p (COMP pa (RAND intruder)) ce
                                  (ENC (SKEY co (RAND intruder)) si)) n))
| nw_fkm211b : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (co:Component;si:Sign)
             (nw s n) -> ~(cfkm211 n co si) ->
             (nw (fkm211 s p pa ce co si) n)
| nw_fkm212a : (s:System;n:Network;p:Principal;ce:Certificate)
               (co1,co2:Component;si:Sign)
             (nw s n) -> (cfkm212 n co1 co2 si) ->
             (nw (fkm212 s p co1 ce co2 si)
             (cons (M2 intruder p co1 ce
                   (ENC (SKEY co2 (RAND intruder)) si)) n))
| nw_fkm212b : (s:System;n:Network;p:Principal;ce:Certificate)
               (co1,co2:Component;si:Sign)
             (nw s n) -> ~(cfkm212 n co1 co2 si) ->
             (nw (fkm212 s p co1 ce co2 si) n)
| nw_fkm221a : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (c:Cipher)
             (nw s n) -> (cfkm221 n c) ->
             (nw (fkm221 s p pa ce c)
             (cons (M2 intruder p (COMP pa (RAND intruder)) ce c) n))
| nw_fkm221b : (s:System;n:Network;p:Principal;pa:Param;ce:Certificate)
               (c:Cipher)
             (nw s n) -> ~(cfkm221 n c) ->
             (nw (fkm221 s p pa ce c) n)
| nw_fkm222a : (s:System;n:Network;p:Principal;co:Component)
               (ce:Certificate;c:Cipher)
             (nw s n) -> (cfkm222 n co c) ->
             (nw (fkm222 s p co ce c) (cons (M2 intruder p co ce c) n))
| nw_fkm222b : (s:System;n:Network;p:Principal;co:Component)
               (ce:Certificate;c:Cipher)
             (nw s n) -> ~(cfkm222 n co c) -> (nw (fkm222 s p co ce c) n)
| nw_fkm31a : (s:System;n:Network;p:Principal;ce:Certificate)
              (co:Component;si:Sign)
             (nw s n) -> (cfkm31 n co si) ->
             (nw (fkm31 s p ce co si)
             (cons (M3 intruder p ce
                   (ENC (SKEY co (RAND intruder)) si)) n))
```

```
| nw_fkm31b : (s:System;n:Network;p:Principal;ce:Certificate)
              (co:Component;si:Sign)
             (nw s n) -> ~(cfkm31 n co si) ->
             (nw (fkm31 s p ce co si) n)
| nw_fkm32a : (s:System;n:Network;p:Principal;ce:Certificate)
              (c:Cipher)
             (nw s n) -> (cfkm32 n c) ->
             (nw (fkm32 s p ce c) (cons (M3 intruder p ce c) n))
| nw_fkm32b : (s:System;n:Network;p:Principal;ce:Certificate)
              (c:Cipher)
             (nw s n) -> ~(cfkm32 n c) -> (nw (fkm32 s p ce c) n).

End STSs.
```

付 録E　検証したい性質の記述(Coq)

```
(* 安全性を示す定理 *)

Theorem inv100 : (s:System)(n:Network;cm:CipherM)
(nw s n) /\ (ccm cm n)
-> (for cm) = intruder.


(* 信頼性を示す定理 *)

Theorem inv200 : (s:System)(n:Network;m,l:nat;sk:Skey)
(nw s n) /\ (In (M4 (P m) (ENCM (P l) sk)) n)
-> (P l) = (key_for sk).

Theorem inv300 : (s:System)(n:Network;m,l:nat;sk:Skey)
(nw s n) /\ (In (M5 (P m) (ENCM (P l) sk)) n)
-> (P l) = (key_for sk).


(* 補題 *)

Lemma inv030 : (s:System)(n:Network;p1,p2:Principal;m:nat)
(co:Component;ce:Certificate;sk:Skey)
(nw s n) /\ (In (M3 p1 p2 ce (ENC sk (SIGN (P m) co))) n)
-> (P m) = (whoz_com co).

Lemma inv020 : (s:System)(n:Network;p1,p2:Principal;m:nat)
(co1,co2:Component;ce:Certificate;sk:Skey)
(nw s n) /\ (In (M2 p1 p2 co1 ce (ENC sk (SIGN (P m) co2))) n)
-> (P m) = (whoz_com co2).

Lemma inv010 : (s:System)(n:Network;sk:Skey;p:Principal;co:Component)
(nw s n) /\ (cci (ENC sk (SIGN p co)) n)
-> p = (whoz_com co).

Lemma inv000 : (s:System)(n:Network;p:Principal;co:Component)
(nw s n) /\ (csi (SIGN p co) n)
-> p = (whoz_com co).

Lemma invm10 : (s:System)(n:Network;m:nat;p:Principal)
(pa:Param;co:Component)
(nw s n) /\ (In (M1 (P m) (P m) p pa co) n)
-> (P m) = (whoz_com co).

Lemma prin_rule : (p:Principal)
{p=intruder}+{(EX m:nat | p=(P m))}.
```

付 録F　表明1の証明譜(Coq)

```
(* --> inv100 *)

Intro.
Induction s.


(* ********************************************** *)
(* (1) Base Case *)

Intros.
Decompose [and] H;Clear H.
Inversion H0.
Rewrite <- H2 in H1;Clear H2.
Inversion_clear H1.



(* (2) Inductive case *)
(* ********************************************** *)
(* (i) (mes1 s p p0 p1) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H3 H4 H5 s0 p2 p3 pa.
Rewrite <- H2 in H1;Clear H2.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.



(* (ii) (mes2 s p p0 p1 c) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 s0 p2 p3 pa co.

(* (ii)-(i) (cmes2 n0 p p0 p1 c) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (ii)-(ii) ~(cmes2 n0 p p0 p1 c) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
```

```
Auto.



(* (iii) (mes3 s p p0 p1 c c0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 H7 s0 p2 p3 pa co1 co2.

(* (iii)-(i) (cmes3 n0 p p0 p1 c c0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (iii)-(ii) ~(cmes3 n0 p p0 p1 c c0) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
Auto.


(* (iv) (mes4 s n p p0 c c0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 H7 s0 m p1 pa co1 co2.

(* (iv)-(i) (cmes4 n1 n p p0 c c0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.

(* Case cm=(ENCM p (SKEY c0 (RAND (P n)))) *)

Simpl.
Unfold rsk in H.
Simpl in H.
Unfold cmes4 in H9.
Decompose [and] H9;Clear H9.
Inversion_clear H1.
Inversion_clear H2.
Cut {p=intruder}+{(EX m:nat|p=(P m))}.
Intro.
Inversion_clear H2.

(* Case p=intruder *)

Auto.

(* Case p=(P x1) *)

Inversion_clear H3.
Rewrite <- H;Clear H.
Rewrite H2.
Rewrite H2 in H1;Clear H2.
Cut (nw s n1) /\
    (In (M2 x (P n) c0 (CERT (P x1) p0) (ENC (SKEY c x0) (SIGN (P x1) c0))) n1).
Intro.
Apply (inv020 s n1 x (P n) x1 c0 c0 (CERT (P x1) p0) (SKEY c x0) H).
```

```
Auto.
Apply (prin_rule p).

(* Case ~cm=(ENCM p (SKEY c0 (RAND (P n)))) *)

Cut (nw s n1) /\ (ccm cm n1).
Intro.
Apply (Hrecs n1 cm H1).
Auto.


(* (iv)-(ii) ~(cmes4 n1 n p p0 c c0) *)

Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H).
Auto.



(* (v) (mes5 s n p p0 c c0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 H7 s0 m p1 pa co1 co2.

(* (v)-(i) (cmes5 n1 n p p0 c c0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.

(* Case cm=(ENCM p (SKEY c0 (RAND (P n)))) *)

Simpl.
Unfold rsk in H.
Simpl in H.

Unfold cmes5 in H9.
Decompose [and] H9;Clear H9.
Inversion_clear H1.
Inversion_clear H2.
Cut {p=intruder}+{(EX m:nat|p=(P m))}.
Intro.
Inversion_clear H2.

(* Case p=intruder *)

Auto.

(* Case p=(P x1) *)

Inversion_clear H3.
Rewrite <- H;Clear H.
Rewrite H2.
Rewrite H2 in H1;Clear H2.
Cut (nw s n1) /\
    (In (M3 x (P n) (CERT (P x1) p0) (ENC (SKEY c x0) (SIGN (P x1) c0))) n1).
Intro.
Apply (inv030 s n1 x (P n) x1 c0 (CERT (P x1) p0) (SKEY c x0) H).
Auto.
Apply (prin_rule p).

(* Case ~cm=(ENCM p (SKEY c0 (RAND (P n)))) *)

Cut (nw s n1) /\ (ccm cm n1).
Intro.
```

```
Apply (Hrecs n1 cm H1).
Auto.


(* (v)-(ii) ~(cmes5 n1 n p p0 c c0) *)

Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H).
Auto.



(* (vi) (fkm11 s p p0 p1 p2) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H3 H4 H5 H6 s0 p3 p4 pa1 pa2.
Rewrite <- H2 in H1;Clear H2.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.



(* (vii) (fkm12 s p p0 p1 c) n) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 s0 p2 p3 pa co.

(* (vii)-(i) (cfkm12 n0 c) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (vii)-(ii) ~(cfkm12 n0 c) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
Auto.



(* (viii) (fkm211 s p p0 c c0 s0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 H7 s1 p1 pa ce co si.

(* (viii)-(i) (cfkm211 n0 c0 s0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.
```

```
(* (viii)-(ii) ~(cfkm211 n0 c0 s0) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
Auto.



(* (viiii) (fkm212 s p c c0 c1 s0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 H7 s1 p0 co1 ce co2 si.

(* (viiii)-(i) (cfkm212 n0 c c1 s0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (viiii)-(ii) ~(cfkm212 n0 c c1 s0) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
Auto.



(* (x) (fkm221 s p p0 c c0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 s0 p1 pa ce c1.

(* (x)-(i) (cfkm221 n0 c0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (x)-(ii) ~(cfkm221 n0 c0) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
Auto.



(* (xi) (fkm222 s p c c0 c1) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 s0 p0 co ce c2.
```

```
(* (xi)-(i) (cfkm222 n c c1) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (xi)-(ii) ~(cfkm222 n c c1) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
Auto.



(* (xii) (fkm31 s p c c0 s0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 H6 s1 p0 ce co si.

(* (xii)-(i) (cfkm31 n0 c0 s0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (xii)-(ii) ~(cfkm31 n0 c0 s0) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
Auto.



(* (xiii) (fkm32 s p c c0) *)

Intros.
Decompose [and] H;Clear H.
Inversion H0;Clear H0 H H2 H4 H5 s0 p0 ce c1.

(* (xiii)-(i) (cfkm32 n0 c0) *)

Rewrite <- H3 in H1;Clear H3.
Inversion_clear H1.
Cut (nw s n0) /\ (ccm cm n0).
Intro.
Apply (Hrecs n0 cm H0).
Auto.


(* (xiii)-(ii) ~(cfkm32 n0 c0) *)

Cut (nw s n) /\ (ccm cm n).
Intro.
Apply (Hrecs n cm H).
```

```
Auto.

Qed.
```

付 録G　Bolignanoの方法によるSTSプロトコルの形式化と検証

```
(*    C = C_K | (C,C) | B   *)
(*    B = K | D | E         *)
(*    K = KS | KI           *)


Require PolyList.

(* データ型の定義 *)

(* メッセージの形式化に必要なデータ型 *)
Parameter D: Set.
Parameter K: Set.
Parameter E: Set.
Inductive B: Set := K2B: K -> B | D2B: D -> B | E2B: E -> B.


(* メッセージの集合 *)
Inductive C: Set := Encrypt: C -> K -> C | Pair: C -> C -> C | B2C: B -> C.


(* 公開情報 *)
Parameter Expo: D -> D -> E.


(* 鍵 *)
Parameter KeyAB: E -> D -> K.  (* セッション鍵 KeyAB *)
Parameter KeyX: D -> K.        (* 秘密鍵 KeyX *)
Parameter inv: K -> K.         (* 復号鍵 *)
Parameters Aid, Bid: D.        (* 主体 A,B の特有の値 *)

(* 秘密鍵の定義 *)
Syntactic Definition Sa := (KeyX Aid).
Syntax constr level 4: SaPP [<<(KeyX Aid)>>] -> ["Sa"].

Syntactic Definition Sb := (KeyX Bid).
Syntax constr level 4: SbPP [<<(KeyX Bid)>>] -> ["Sb"].

(* 公開鍵の定義 *)
Syntactic Definition Pa := (inv (KeyX Aid)).
Syntax constr level 4: PaPP [<<(inv (KeyX Aid))>>] -> ["Pa"].

Syntactic Definition Pb := (inv (KeyX Bid)).
Syntax constr level 4: PbPP [<<(inv (KeyX Bid))>>] -> ["Pb"].


(* 侵入者の振る舞いの定義 *)

(* 侵入者がネットワークから手に入れることのできる情報空間 *)
Syntactic Definition SS := (list C).
Syntax constr level 4: SPP [<<(list C)>>] -> ["SS"].
```

```
(* 取得済みの情報かどうかの判定 *)

Inductive known_in: C -> SS -> Prop :=
  E0: (c1, c2: C) (s: SS) (known_in c1 s) -> (known_in c1 (cons c2 s))
| EP0: (c: C) (s1, s2: SS) (known_in c s1) -> (known_in c (app s1 s2))
| A1: (c1, c2: C) (s: SS)
      (known_in c1 s) -> (known_in c2 s) -> (known_in (Pair c1 c2) s)
| A2A: (c1, c2: C) (s: SS) (known_in (Pair c1 c2) s) -> (known_in c1 s)
| A2B: (c1, c2: C) (s: SS) (known_in (Pair c1 c2) s) -> (known_in c2 s)
| A3: (c1: C) (k1: K) (s: SS)
      ((known_in c1 s) -> (known_in (B2C (K2B k1)) s) ->
      (known_in (Encrypt c1 k1) s))
| A4: (c1: C) (k1: K) (s: SS)
      (known_in (Encrypt c1 k1) s) -> (known_in (B2C (K2B (inv k1))) s) ->
      (known_in c1 s)
| EX1: (d1, d2: D) (s: SS)
       (known_in (B2C (D2B d1)) s) -> ~d2=Aid /\ ~d2=Bid ->
       (known_in (B2C (E2B (Expo d1 d2))) s)
| EX2: (d1: D) (e1: E) (s: SS)
       (known_in (B2C (E2B e1)) s) -> ~d1=Aid /\ ~d1=Bid ->
       (known_in (B2C (K2B (KeyAB e1 d1))) s).


(* 状態空間の定義 *)

(* A(B) の状態 *)
(* メッセージの送信、受信で A(B) が知りえる情報 , 保持すべき情報 *)
(* パラメータ A , 公開値 EXb(EXa) , 鍵交換を行った相手 W *)

Inductive AState: Set :=
 APEXbW : D -> E -> D -> AState.

Inductive BState: Set :=
 BPEXaW : D -> E -> D -> BState.

(* メッセージ空間の状態 *)
Inductive GlobalState: Set :=
  ABI: AState -> BState -> SS -> GlobalState.


(* メッセージ送信における状態空間の変化 *)

Definition Triple := [c1, c2, c3: C] (Pair c1 (Pair c2 c3)).

(* メッセージ 1 の送信による状態変化 *)

Definition rel1 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases sta1 of (APEXbW p1 exb1 w1) =>
   s2=(cons (Pair (B2C (D2B p1)) (B2C (E2B (Expo p1 Aid)))) s1) /\
   sta1 = sta2 /\ stb1 = stb2
end
end.


(* メッセージ 1 の受信による資源の変化 *)

Definition rel2 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases stb1 stb2 of (BPEXaW _ _ w1)
                     (BPEXaW p2 exa2 w2) =>
   (known_in (Pair (B2C (D2B p2)) (B2C (E2B exa2))) s1) /\
   sta1 = sta2 /\ s1 = s2 /\
   w1 = w2
end
end.
```

```
(* メッセージ２の送信による状態変化 *)

Definition rel3 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases stb1 of (BPEXaW p1 exa1 _) =>
   s2=(cons (Triple (B2C (E2B (Expo p1 Bid))) (B2C (K2B Pb))
                    (Encrypt
                    (Encrypt (Pair (B2C (E2B exa1))
                             (B2C (E2B (Expo p1 Bid)))) Sb)
                    (KeyAB exa1 Bid))) s1) /\
   sta1 = sta2 /\ stb1 = stb2
end
end.


(* メッセージ２の受信による資源の変化 *)

Definition rel4 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases sta1 sta2 of (APEXbW p1 _  _)
                     (APEXbW p2 exb2 w2) =>
   (known_in (Triple (B2C (E2B exb2)) (B2C (K2B Pb))
                     (Encrypt
                     (Encrypt (Pair (B2C (E2B (Expo p2 Aid)))
                                    (B2C (E2B exb2))) Sb)
                     (KeyAB (Expo p2 Aid) w2))) s1) /\
   stb1 = stb2 /\ s1 = s2 /\
   p1 = p2
end
end.


(* メッセージ３の送信による状態変化 *)

Definition rel5 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases sta1 of (APEXbW p1 exb1 w1) =>
   s2 = (cons (Pair (B2C (K2B Pa))
                    (Encrypt
                    (Encrypt (Pair (B2C (E2B (Expo p1 Aid)))
                                   (B2C (E2B exb1))) Sa)
                    (KeyAB exb1 Aid))) s1) /\
  sta1 = sta2 /\ stb1 = stb2
end
end.


(* メッセージ３の受信による資源の変化 *)

Definition rel6 := [st1, st2: GlobalState]
  Cases st1 st2 of (ABI sta1 stb1 s1) (ABI sta2 stb2 s2) =>
  Cases stb1 stb2 of (BPEXaW p1 exa1 _)
                     (BPEXaW p2 exa2 w2) =>
  (known_in (Pair (B2C (K2B Pa))
                  (Encrypt
                  (Encrypt (Pair (B2C (E2B exa2))
                                 (B2C (E2B (Expo p2 Bid)))) Sa)
                  (KeyAB (Expo p2 Aid) w2))) s1) /\
  sta1 = sta2 /\ s1 = s2 /\
  p1 = p2 /\ exa1 = exa2
end
end.
```

```
(* メッセージの送受信 *)

Definition rel := [st1, st2: GlobalState]
  (rel1 st1 st2) \/ (rel2 st1 st2) \/ (rel3 st1 st2) \/ (rel4 st1 st2) \/
  (rel5 st1 st2) \/ (rel6 st1 st2).



(*  性質を表す述語 *)

(* 交換した公開値は相手のものである *)

Definition CipherM1 := [st: GlobalState]
Cases st of (ABI (APEXbW p exb _) _ _) => exb = (Expo p Bid)
end.

Definition CipherM2 := [st: GlobalState]
Cases st of (ABI _ (BPEXaW p exa _) _) => exa = (Expo p Aid)
end.


(* メッセージの送受信により、性質 CipherM1,M2 は失われない *)

Theorem inv100: (st1:GlobalState)(st2:GlobalState)
(CipherM1 st1) -> (rel st1 st2) -> (CipherM1 st2).

Proof.
Induction st1; Intros a b l; Elim a; Elim b;
Induction st2; Intros a0 b0 l0; Elim a0; Elim b0.
Do 6 Intro.
Unfold CipherM1; Unfold rel; Intros expo_bid rel1_6.

(* rel1 *)
Elim rel1_6. Intros r1.
Elim r1. Intros eq_l0 and1; Elim and1; Intros H t.
Clear rel1_6 r1 eq_l0 and1 t.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel2 *)
Intro rel2_6; Elim rel2_6. Intros r2.
Elim r2; Intros know_d3e1_l and1; Elim and1; Intros H and2.
Clear rel1_6 rel2_6 r2 know_d3e1_l and1 and2.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel3 *)
Intros rel3_6; Elim rel3_6. Intros r3.
Elim r3; Intros eq_l0 and1; Elim and1; Intros H t.
Clear rel1_6 rel2_6 rel3_6 r3 eq_l0 and1 t.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel4 *)
Intros rel4_6; Elim rel4_6. Intros r4.
Elim r4; Intros know_c1_l and1; Elim and1; Intros t1 and2; Elim and2.
Intros t2 t3. Clear rel1_6 rel2_6 rel3_6 rel4_6 r4 and1 and2 t1 t2 t3.
Elim know_c1_l; Intros; Assumption.

(* rel5 *)
Intros rel5_6; Elim rel5_6. Intros r5.
Elim r5; Intros eq_l0 and1; Elim and1; Intros H t.
Clear rel1_6 rel2_6 rel3_6 rel4_6 rel5_6 r5 eq_l0 and1 t.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel6 *)
Intros rel6_6; Elim rel6_6.
Intros know_c1_l and1; Elim and1; Intros H and2.
Clear rel1_6 rel2_6 rel3_6 rel4_6 rel5_6 rel6_6 know_c1_l and1 and2.
```

```
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

Qed.


Theorem inv200: (st1:GlobalState)(st2:GlobalState)
(CipherM2 st1) -> (rel st1 st2) -> (CipherM2 st2).

Proof.
Induction st1; Intros a b l; Elim a; Elim b;
Induction st2; Intros a0 b0 l0; Elim a0; Elim b0.
Do 6 Intro.
Unfold CipherM2; Unfold rel; Intros expo_bid rel1_6.

(* rel1 *)
Elim rel1_6. Intros r1.
Elim r1. Intros eq_l0 and1; Elim and1; Intros t H.
Clear rel1_6 r1 eq_l0 and1 t.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel2 *)
Intro rel2_6; Elim rel2_6. Intros r2.
Elim r2; Intros know_d3e1_l and1.
Clear rel1_6 rel2_6 r2 and1.
Elim know_d3e1_l; Intros; Assumption.

(* rel3 *)
Intros rel3_6; Elim rel3_6. Intros r3.
Elim r3; Intros eq_l0 and1; Elim and1; Intros t H.
Clear rel1_6 rel2_6 rel3_6 r3 eq_l0 and1 t.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel4 *)
Intros rel4_6; Elim rel4_6. Intros r4.
Elim r4; Intros know_c1_l and1; Elim and1; Intros H and2.
Clear rel1_6 rel2_6 rel3_6 rel4_6 r4 and1 and2.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel5 *)
Intros rel5_6; Elim rel5_6. Intros r5.
Elim r5; Intros eq_l0 and1; Elim and1; Intros t H.
Clear rel1_6 rel2_6 rel3_6 rel4_6 rel5_6 r5 eq_l0 and1 t.
Inversion H; Rewrite <- H1; Rewrite <- H2; Assumption.

(* rel6 *)
Intros rel6_6; Elim rel6_6.
Intros know_c1_l and1; Elim and1; Intros H and2.
Clear rel1_6 rel2_6 rel3_6 rel4_6 rel5_6 rel6_6 and1 and2.
Elim know_c1_l; Intros; Assumption.

Qed.
```

付 録H　抽出したプログラムfun_nw

```
type nat =
  | O
  | S of nat

type 'a sig0 = 'a
  (* singleton inductive, whose constructor was exist *)

type sumbool =
  | Left
  | Right

type 'a list =
  | Nil
  | Cons of 'a * 'a list

type principal = nat
  (* singleton inductive, whose constructor was P *)

type param = nat
  (* singleton inductive, whose constructor was PA *)

type randum = principal
  (* singleton inductive, whose constructor was RAND *)

type component =
  | COMP of param * randum

type skey =
  | SKEY of component * randum

type certificate =
  | CERT of principal * param

type sign =
  | SIGN of principal * component

type cipher =
  | ENC of skey * sign

type cipherM =
  | ENCM of principal * skey

type msg =
  | M1 of principal * principal * principal * param * component
  | M2 of principal * principal * component * certificate * cipher
  | M3 of principal * principal * certificate * cipher
  | M4 of principal * cipherM
  | M5 of principal * cipherM

type network = msg list

type system =
  | Init
  | Mes1 of system * principal * principal * param
```

```
  | Mes2 of system * principal * principal * param * component
  | Mes3 of system * principal * principal * param * component * component
  | Mes4 of system * nat * principal * param * component * component
  | Mes5 of system * nat * principal * param * component * component

(** val eq_nat : nat -> nat -> sumbool **)

let rec eq_nat n m =
  match n with
    | O -> (match m with
              | O -> Left
              | S n0 -> Right)
    | S n0 -> (match m with
                 | O -> Right
                 | S n1 -> eq_nat n0 n1)

(** val eq_Prin : principal -> principal -> sumbool **)

let eq_Prin p1 p2 =
  eq_nat p1 p2

(** val eq_Para : param -> param -> sumbool **)

let eq_Para p1 p2 =
  eq_nat p1 p2

(** val eq_Rand : randum -> randum -> sumbool **)

let eq_Rand r1 r2 =
  eq_Prin r1 r2

(** val eq_Comp : component -> component -> sumbool **)

let eq_Comp c1 c2 =
  let COMP (x, x0) = c1 in
  let COMP (p0, x1) = c2 in
  (match eq_Para x p0 with
     | Left -> eq_Rand x0 x1
     | Right -> Right)

(** val eq_Cert : certificate -> certificate -> sumbool **)

let eq_Cert c1 c2 =
  let CERT (x, x0) = c1 in
  let CERT (p1, x1) = c2 in
  (match eq_Prin x p1 with
     | Left -> eq_Para x0 x1
     | Right -> Right)

(** val eq_Skey : skey -> skey -> sumbool **)

let eq_Skey s1 s2 =
  let SKEY (x, x0) = s1 in
  let SKEY (c0, x1) = s2 in
  (match eq_Comp x c0 with
     | Left -> eq_Rand x0 x1
     | Right -> Right)

(** val eq_Sign : sign -> sign -> sumbool **)

let eq_Sign s1 s2 =
  let SIGN (x, x0) = s1 in
  let SIGN (p0, x1) = s2 in
  (match eq_Prin x p0 with
     | Left -> eq_Comp x0 x1
     | Right -> Right)
```

```
(** val eq_Ciph : cipher -> cipher -> sumbool **)

let eq_Ciph c1 c2 =
  let ENC (x, x0) = c1 in
  let ENC (s1, x1) = c2 in
  (match eq_Skey x s1 with
     | Left -> eq_Sign x0 x1
     | Right -> Right)

(** val eq_CipM : cipherM -> cipherM -> sumbool **)

let eq_CipM c1 c2 =
  let ENCM (x, x0) = c1 in
  let ENCM (p0, x1) = c2 in
  (match eq_Prin x p0 with
     | Left -> eq_Skey x0 x1
     | Right -> Right)

(** val eq_Msg : msg -> msg -> sumbool **)

let eq_Msg m1 m2 =
  match m1 with
    | M1 (x, x0, x1, x2, x3) ->
        (match m2 with
           | M1 (p3, x4, x5, x6, x7) ->
               (match eq_Prin x p3 with
                  | Left ->
                      (match eq_Prin x0 x4 with
                         | Left ->
                             (match eq_Prin x1 x5 with
                                | Left ->
                                    (match eq_Para x2 x6 with
                                       | Left -> eq_Comp x3 x7
                                       | Right -> Right)
                                | Right -> Right)
                         | Right -> Right)
                  | Right -> Right)
           | M2 (p3, p4, c0, c1, c2) -> Right
           | M3 (p3, p4, c0, c1) -> Right
           | M4 (p3, c0) -> Right
           | M5 (p3, c0) -> Right)
    | M2 (x, x0, x1, x2, x3) ->
        (match m2 with
           | M1 (p1, p2, p3, p4, c2) -> Right
           | M2 (p1, x4, x5, x6, x7) ->
               (match eq_Prin x p1 with
                  | Left ->
                      (match eq_Prin x0 x4 with
                         | Left ->
                             (match eq_Comp x1 x5 with
                                | Left ->
                                    (match eq_Cert x2 x6 with
                                       | Left -> eq_Ciph x3 x7
                                       | Right -> Right)
                                | Right -> Right)
                         | Right -> Right)
                  | Right -> Right)
           | M3 (p1, p2, c2, c3) -> Right
           | M4 (p1, c2) -> Right
           | M5 (p1, c2) -> Right)
    | M3 (x, x0, x1, x2) ->
        (match m2 with
           | M1 (p1, p2, p3, p4, c1) -> Right
           | M2 (p1, p2, c1, c2, c3) -> Right
           | M3 (p1, x3, x4, x5) ->
```

```
              (match eq_Prin x p1 with
                 | Left ->
                     (match eq_Prin x0 x3 with
                        | Left ->
                            (match eq_Cert x1 x4 with
                               | Left -> eq_Ciph x2 x5
                               | Right -> Right)
                        | Right -> Right)
                 | Right -> Right)
          | M4 (p1, c1) -> Right
          | M5 (p1, c1) -> Right)
    | M4 (x, x0) ->
        (match m2 with
           | M1 (p0, p1, p2, p3, c0) -> Right
           | M2 (p0, p1, c0, c1, c2) -> Right
           | M3 (p0, p1, c0, c1) -> Right
           | M4 (p0, x1) ->
               (match eq_Prin x p0 with
                  | Left -> eq_CipM x0 x1
                  | Right -> Right)
           | M5 (p0, c0) -> Right)
    | M5 (x, x0) ->
        (match m2 with
           | M1 (p0, p1, p2, p3, c0) -> Right
           | M2 (p0, p1, c0, c1, c2) -> Right
           | M3 (p0, p1, c0, c1) -> Right
           | M4 (p0, c0) -> Right
           | M5 (p0, x1) ->
               (match eq_Prin x p0 with
                  | Left -> eq_CipM x0 x1
                  | Right -> Right))

(** val cmes5_sub3 : network -> principal -> principal -> certificate ->
                     component -> cipher -> sumbool **)

let rec cmes5_sub3 n p1 p2 ce co ci =
  match n with
    | Nil -> Right
    | Cons (a, l) ->
        (match eq_Msg a (M2 (p1, p2, co, ce, ci)) with
           | Left -> Left
           | Right -> cmes5_sub3 l p1 p2 ce co ci)

(** val cmes5_sub2 : msg -> principal -> certificate -> component -> sign ->
                     sumbool **)

let cmes5_sub2 m p ce co si =
  match m with
    | M1 (x, x0, x1, x2, x3) -> Right
    | M2 (x, x0, x1, x2, x3) -> Right
    | M3 (x, x0, x1, x2) ->
        (match eq_Cert x1 ce with
           | Left ->
               (match eq_Prin x0 p with
                  | Left ->
                      let ENC (x3, x4) = x2 in
                      let SKEY (x5, x6) = x3 in
                      (match eq_Sign x4 si with
                         | Left -> eq_Comp x5 co
                         | Right -> Right)
                  | Right -> Right)
           | Right -> Right)
    | M4 (x, x0) -> Right
    | M5 (x, x0) -> Right

(** val cmes5_sub1 : network -> principal -> certificate -> component -> sign
```

```
                    -> sumbool **)

let rec cmes5_sub1 l p ce co si =
  match l with
    | Nil -> Right
    | Cons (a, l0) ->
        (match cmes5_sub1 l0 p ce co si with
           | Left -> Left
           | Right -> cmes5_sub2 a p ce co si)

(** val fun_cmes5 : network -> principal -> principal -> param -> component
                    -> component -> sumbool **)

let fun_cmes5 n p1 p2 pa co1 co2 =
  match cmes5_sub1 n p1 (CERT (p2, pa)) co1 (SIGN (p2, co2)) with
    | Left ->
        cmes5_sub3 n p1 p2 (CERT (m, pa)) co1 (ENC ((SKEY (co2, m)),
                                               (SIGN (p1,co1))))
    | Right -> Right

(** val cmes4_sub3 : network -> principal -> principal -> principal -> param
                    -> component -> sumbool **)

let rec cmes4_sub3 n p1 p2 p3 pa co =
  match n with
    | Nil -> Right
    | Cons (a, l) ->
        (match eq_Msg a (M1 (p1, p2, p3, pa, co)) with
           | Left -> Left
           | Right -> cmes4_sub3 l p1 p2 p3 pa co)

(** val cmes4_sub2 : msg -> principal -> certificate -> component ->
                    component -> sign -> sumbool **)

let cmes4_sub2 m p ce co1 co2 si =
  match m with
    | M1 (x, x0, x1, x2, x3) -> Right
    | M2 (x, x0, x1, x2, x3) ->
        (match eq_Cert x2 ce with
           | Left ->
               (match eq_Comp x1 co2 with
                  | Left ->
                      (match eq_Prin x0 p with
                         | Left ->
                             let ENC (x4, x5) = x3 in
                             let SKEY (x6, x7) = x4 in
                             (match eq_Sign x5 si with
                                | Left -> eq_Comp x6 co1
                                | Right -> Right)
                         | Right -> Right)
                  | Right -> Right)
           | Right -> Right)
    | M3 (x, x0, x1, x2) -> Right
    | M4 (x, x0) -> Right
    | M5 (x, x0) -> Right

(** val cmes4_sub1 : network -> principal -> certificate -> component ->
                    component -> sign -> sumbool **)

let rec cmes4_sub1 l p ce co1 co2 si =
  match l with
    | Nil -> Right
    | Cons (a, l0) ->
        (match cmes4_sub1 l0 p ce co1 co2 si with
           | Left -> Left
           | Right -> cmes4_sub2 a p ce co1 co2 si)
```

```
(** val fun_cmes4 : network -> principal -> principal -> param -> component
                   -> component -> sumbool **)

let fun_cmes4 n p1 p2 pa co1 co2 =
  match cmes4_sub1 n p1 (CERT (p2, pa)) co1 co2 (SIGN (p2, co2)) with
    | Left -> cmes4_sub3 n p1 p1 p2 pa co1
    | Right -> Right

(** val cmes3_sub3 : network -> principal -> principal -> principal -> param
                     -> component -> sumbool **)

let rec cmes3_sub3 n p1 p2 p3 pa co =
  match n with
    | Nil -> Right
    | Cons (a, l) ->
        (match eq_Msg a (M1 (p1, p2, p3, pa, co)) with
           | Left -> Left
           | Right -> cmes3_sub3 l p1 p2 p3 pa co)

(** val cmes3_sub2 : msg -> principal -> certificate -> component ->
                     component -> sign -> sumbool **)

let cmes3_sub2 m p ce co1 co2 si =
  match m with
    | M1 (x, x0, x1, x2, x3) -> Right
    | M2 (x, x0, x1, x2, x3) ->
        (match eq_Cert x2 ce with
           | Left ->
               (match eq_Comp x1 co2 with
                  | Left ->
                      (match eq_Prin x0 p with
                         | Left ->
                             let ENC (x4, x5) = x3 in
                             let SKEY (x6, x7) = x4 in
                             (match eq_Sign x5 si with
                                | Left -> eq_Comp x6 co1
                                | Right -> Right)
                         | Right -> Right)
                  | Right -> Right)
           | Right -> Right)
    | M3 (x, x0, x1, x2) -> Right
    | M4 (x, x0) -> Right
    | M5 (x, x0) -> Right

(** val cmes3_sub1 : network -> principal -> certificate -> component ->
                     component -> sign -> sumbool **)

let rec cmes3_sub1 l p ce co1 co2 si =
  match l with
    | Nil -> Right
    | Cons (a, l0) ->
        (match cmes3_sub1 l0 p ce co1 co2 si with
           | Left -> Left
           | Right -> cmes3_sub2 a p ce co1 co2 si)

(** val fun_cmes3 : network -> principal -> principal -> param -> component
                    -> component -> sumbool **)

let fun_cmes3 n p1 p2 pa co1 co2 =
  match cmes3_sub1 n p1 (CERT (p2, pa)) co1 co2 (SIGN (p2, co2)) with
    | Left -> cmes3_sub3 n p1 p1 p2 pa co1
    | Right -> Right

(** val cmes2_sub2 : msg -> principal -> principal -> param -> component ->
                     sumbool **)
```

```
let cmes2_sub2 m p1 p2 pa co =
  match m with
    | M1 (x, x0, x1, x2, x3) ->
        (match eq_Comp x3 co with
           | Left ->
               (match eq_Para x2 pa with
                  | Left ->
                      (match eq_Prin x1 p2 with
                         | Left -> eq_Prin x0 p1
                         | Right -> Right)
                  | Right -> Right)
           | Right -> Right)
    | M2 (x, x0, x1, x2, x3) -> Right
    | M3 (x, x0, x1, x2) -> Right
    | M4 (x, x0) -> Right
    | M5 (x, x0) -> Right

(** val cmes2_sub1 : network -> principal -> principal -> param -> component
                     -> sumbool **)

let rec cmes2_sub1 l p1 p2 pa co =
  match l with
    | Nil -> Right
    | Cons (a, l0) ->
        (match cmes2_sub1 l0 p1 p2 pa co with
           | Left -> Left
           | Right -> cmes2_sub2 a p1 p2 pa co)

(** val fun_cmes2 : network -> principal -> principal -> param -> component
                    -> sumbool **)

let fun_cmes2 n p1 p2 pa co =
  cmes2_sub1 n p1 p2 pa co

(** val fun_nw : system -> network sig0 **)

let rec fun_nw = function
  | Init -> Nil
  | Mes1 (s0, p, p0, p1) -> Cons ((M1 (p, p, p0, p1, (COMP (p1, p)))),
      (fun_nw s0))
  | Mes2 (s0, p, p0, p1, c) ->
      let hrecs = fun_nw s0 in
      (match fun_cmes2 hrecs p p0 p1 c with
         | Left -> Cons ((M2 (p0, p, (COMP (p1, p0)), (CERT (p0, p1)), (ENC
             ((SKEY (c, p0)), (SIGN (p0, (COMP (p1, p0)))))))), hrecs)
         | Right -> hrecs)
  | Mes3 (s0, p, p0, p1, c, c0) ->
      let hrecs = fun_nw s0 in
      (match fun_cmes3 hrecs p p0 p1 c c0 with
         | Left -> Cons ((M3 (p, p0, (CERT (p, p1)), (ENC ((SKEY (c0, p)),
             (SIGN (p, c)))))), hrecs)
         | Right -> hrecs)
  | Mes4 (s0, n, p, p0, c, c0) ->
      let hrecs = fun_nw s0 in
      (match fun_cmes4 hrecs n p p0 c c0 with
         | Left -> Cons ((M4 (n, (ENCM (p, (SKEY (c0, n)))))), hrecs)
         | Right -> hrecs)
  | Mes5 (s0, n, p, p0, c, c0) ->
      let hrecs = fun_nw s0 in
      (match fun_cmes5 hrecs n p p0 c c0 with
         | Left -> Cons ((M5 (n, (ENCM (p, (SKEY (c0, n)))))), hrecs)
         | Right -> hrecs)
```